INSTITUTO
UNIVERSITÁRIO
DE LISBOA

UNIVERSIDADE
DE LISBOA

Avaliação de opções através de métodos de *machine learning*

Raquel Lopes Coutinho

Mestrado em Matemática Financeira

Orientadores:
Professor Doutor José Carlos Gonçalves Dias, Professor Catedrático,
ISCTE Business School

Professor Doutor João Pedro Bento Ruas, Professor Auxiliar,
ISCTE Business School

Setembro, 2023

Departamento de Finanças

Departamento de Matemática

Avaliação de opções através de métodos de *machine learning*

Raquel Lopes Coutinho

Mestrado em Matemática Financeira

Orientadores:
Professor Doutor José Carlos Gonçalves Dias, Professor Catedrático,
ISCTE Business School

Professor Doutor João Pedro Bento Ruas, Professor Auxiliar,
ISCTE Business School

Setembro, 2023

Dedicado aos meus pais por todo o apoio e paciência, principalmente nos momentos de adversidade.

Agradecimento

O meu mais sincero agradecimento aos Professores José Carlos Dias e João Pedro Ruas pelo aconselhamento, disponibilidade, ajuda e orientação que me ofereceram durante o desenvolvimento desta dissertação.

Agradeço aos meus pais, pelo apoio, motivação e força que me deram ao longo desta etapa. Sem eles este projeto não teria sido possível.

Por último, agradeço aos meus amigos pelo cuidado, incentivo e por me apoiarem nesta jornada tão importante para mim.

Resumo

Esta dissertação estuda a performance dos métodos de *machine learning* mais populares na avaliação do preço de opções. Os métodos explorados nesta dissertação são o *Support Vector Regression*, o *Random Forest*, o XGBoost, o LightGBM e o *Neural Networks*. De forma a existir um elo de comparação, estudou-se de forma equivalente os modelos teóricos Black-Scholes-Merton e Corrado e Su aplicando tanto parâmetros históricos como implícitos. A análise foi realizada em opções de compra europeias e mensais do índice S&P500. Esta avaliação teve por base o artigo [**1**].

Abstract

This dissertation investigates the performance of the most popular machine learning methods in the evaluation of options prices. The methods explored in this study encompass Support Vector Regression, Random Forest, XGBoost, LightGBM and Neural Networks. To establish a basis for comparison, an equivalent study of the theoretical models, Black-Scholes-Merton and Corrado and Su, was conducted, incorporating both historical and implied parameters. The analysis focused on European monthly call options of the S&P500 index. This evaluation was founded upon the article by [**1**].

Índice

Agradecimento iii

Resumo v

Abstract vii

Capítulo 1. Introdução 1
1.1. Support Vector Regression 5
1.2. Random Forest 8
1.3. XGBoost 9
1.4. LightGBM 11
1.5. Neural Networks 12

Capítulo 2. Dados e Metodologia 13
2.1. Implementação do modelo de Black-Scholes-Merton 14
2.2. Implementação do modelo de Corrado e Su 15
2.3. Volatilidade Histórica 16
2.4. Implementação do método Support Vector Regression 16
2.5. Implementação do método Random Forest 17
2.6. Implementação do método XGBoost e do método LightGBM 17
2.7. Implementação do método Neural Networks 18

Capítulo 3. Resultados 19

Capítulo 4. Conclusões 23

Referências Bibliográficas 25

Apêndice A. Códigos em Python 27
A.1. Tratamento dos Dados 27
A.2. Volatilidade Histórica 28
A.3. Volatilidade Implícita - Black-Scholes-Merton 29
A.4. Volatilidade Implícita, Skewness e Kurtosis - Corrado e Su 31
A.5. Modelo de Black-Scholes-Merton 32
A.6. Modelo de Corrado e Su 33
A.7. Método Support Vector Regression 33
A.8. Método Random Forest 34

A.9. Método XGBoost 34
A.10. Método LightGBM 34
A.11. Método Neural Networks 34

1

Introdução

Um instrumento derivado, ou simplesmente derivado, é um instrumento financeiro resultante de um outro ativo, o ativo subjacente. Ou seja, cada instrumento derivado está ligado a outro ativo financeiro do qual não pode ser separado. Assim, o valor do derivado provém do valor do ativo financeiro subjacente que, por sua vez, advém do preço de mercado a que está associado.

Os derivados são os instrumentos financeiros mais utilizados por investidores como cobertura dos riscos referentes à economia (*hedging*), especulação e arbitragem (*arbitrage*).

Ao contrário das ações, obrigações e instrumentos monetários que são o mecanismo de financiamento direto das empresas, os derivados são um meio de salvaguarda das empresas face ao risco de mercado (como oscilações das taxas de câmbio, das taxas de juro, das cotações bolsistas, da inflação, etc.), face ao risco de crédito (como o incumprimento, a insolvência e a iliquidez do devedor) e face a outros riscos económicos.

Por isso, se uma empresa necessita regularmente de uma determina matéria-prima que é sujeita a uma enorme volatilidade de preço no mercado, como o combustível, pode prevenir-se contra subidas exorbitantes do preço através da compra de futuros que fixam hoje o preço a pagar no futuro. Ou se um comerciante europeu, titular de créditos em dólares, temer uma evolução desfavorável da taxa de câmbio euro/dólar, pode prevenir-se contra esse risco comprando um *swap* de divisas. Ou se um banqueiro que prevê adquirir a prazo uma carteira de ações pode prevenir-se dos riscos correspondentes às descidas das respetivas cotações através da compra de uma opção de venda sobre esses valores. Estes são alguns casos onde os derivados são utilizados como cobertura de risco.

No entanto, os derivados também são utilizados para fins de especulação, como permitir ao investidor realizar aplicações de modo a obter um lucro. Ou seja, se um empresário acredita que o preço da prata irá subir nos próximos seis meses, então irá adquirir um futuro para entregar 1000 onças de prata daqui a seis meses, no entanto ao preço de hoje. Se, após seis meses, o preço da prata aumentar, então o empresário poderá vender as 1000 onças com lucro.

A finalidade da arbitragem é igualmente a de obter o maior lucro possível, no entanto tirando partido das imperfeições dos mercados ou dos preços dos ativos subjacentes. Caso uma ação indique um preço futuro diferente do que os investidores do mercado à vista

projetam, a estratégia é comprar o ativo no mercado em que o preço está mais baixo e vender naquele em que está mais alto. O lucro de cada operação costuma ser pequeno e por isso, os ganhos vêm da quantidade e do volume de compras e vendas.

Existem quatro tipos de derivados: os futuros, os *forwards*, os *swaps* e as opções.

Os futuros são contratos financeiros que estabelecem a compra ou venda de um ativo financeiro em uma data futura específica, a um preço determinado no momento da negociação. Esses contratos são negociados em bolsas de valores ou mercados futuros e são utilizados para cobertura de risco ou especulação. Ao entrar em um contrato de futuro, as partes concordam em comprar ou vender o ativo subjacente na data de vencimento do contrato, independentemente das alterações de preço que ocorrerem durante o período. O comprador do futuro tem a obrigação de comprar o ativo subjacente.

Os *forwards* são contratos financeiros que estabelecem a compra ou venda de um ativo financeiro em uma data futura específica, a um preço determinado no momento da negociação. Ao contrário dos futuros, os *forwards* são acordos privados entre as partes e por isso não são negociados em bolsas de valores, o que torna este tipo de derivado mais arriscado.

Os swaps são contratos financeiros em que duas partes concordam em trocar fluxos de pagamentos futuros, de acordo com um determinado critério pré-estabelecido. Essas transações são amplamente utilizadas para riscos financeiros e ajustar a estrutura dos passivos e ativos de uma empresa. Os *swaps* mais comuns são os *swaps* de taxa de juros, em que uma parte paga uma taxa de juros fixa e recebe uma taxa de juros variável, ou vice-versa.

Os *swaps* são negociados no mercado de balcão (*over-the-counter*), o que significa que são acordos feitos diretamente entre as partes envolvidas, sem passar por uma bolsa de valores ou mercado organizado.

As opções são instrumentos financeiros que concedem ao titular o direito, mas não a obrigação, de comprar (opção de compra) ou vender (opção de venda) um ativo subjacente a um preço pré-estabelecido (preço de exercício) em uma data futura. As opções oferecem aos investidores a oportunidade de se beneficiarem das variações dos preços dos ativos subjacentes, sem a necessidade de possuí-los efetivamente. Ao adquirir uma opção, o titular tem o direito de exercê-la ou vendê-la no mercado antes do vencimento, se considerar vantajoso.

Existem dois tipos de opções: as opções de compra e as opções de venda. Cada um desse tipo de opções podem ser opções de estilo europeu ou de estilo americano. As opções de compra de estilo europeu (*European call options*) dão ao titular o direito de comprar o ativo subjacente ao preço de exercício na data de vencimento, enquanto as opções de compra de estilo americano (*American call options*) dão ao titular o direito de comprar o ativo subjacente ao preço de exercício a qualquer momento até à data de vencimento. O titular da opção de compra espera que o preço do ativo subjacente aumente, pois isso vai permitir-lhe comprar o ativo por um preço inferior ao valor de mercado atual. As opções

de venda de estilo europeu (*European put options*) dão ao titular o direito de vender o ativo subjacente ao preço de exercício na data de vencimento, enquanto as opções de venda de estilo americano (*American put options*) dão ao titular o direito de vender o ativo subjacente ao preço de exercício a qualquer momento até à data de vencimento. O titular da opção de venda espera que o preço do ativo subjacente caia, pois isso vai permitir-lhe vender o ativo por um preço superior ao valor de mercado atual.

Todas as opções possuem um prémio, que é o valor pago pelo titular ao vendedor da opção para adquirir esse direito. Ao adquirir uma opção, o titular não é obrigado a exercê-la. Se o preço do ativo subjacente não for favorável, o titular pode optar por não exercer a opção e perderá apenas o valor do prémio pago inicialmente.

Para além de todas estas vantagens que existem na compra ou venda de derivados, existem desvantagens que precisam de ser levadas em consideração. Entre elas, os riscos significativos, como a possibilidade de enormes perdas (a complexidade dos instrumentos financeiros, o desconhecimento ou uso inadequado dos derivados pode levar a perdas significativas), volatilidade e liquidez, alguns derivados podem ser altamente voláteis, o que significa que seus preços podem alterar-se rapidamente e, para além disso, nem todos os derivados possuem alta liquidez, o que pode dificultar as negociações no momento desejado.

Assim, não é surpreendente que as perdas significativas relacionadas aos derivados sejam cada vez maiores. De todos os derivados, o mais conhecido e utilizado no mercado financeiro são as opções. Esta fama deve-se ao modelo estocástico de Black-Scholes, criado para ser possível calcular o preço de mercado de uma opção. Este modelo foi publicado por Fischer Black e Myron Scholes num artigo intitulado "The Pricing of Options and Corporate Liabilities"em 1973. O modelo de Black-Scholes supõe que é possível emprestar e pedir emprestado a uma taxa de juros conhecida e contante, que o preço segue um movimento Browniano geométrico com tendência e volatilidade constantes, que não há custos de transação e que o ativo não paga dividendos. O modelo só se aplica a opções europeias, no entanto pode ser estendido para opções americanas e para opções que pagam dividendos se estes forem uma proporção conhecida do preço da opção. No entanto, nenhuma destas suposições mostra a veracidade das condições de um mercado financeiro. Normalmente a taxa de juro não é constante, a volatilidade não é constante e paga-se sempre custos de transação, por muito pequenos que sejam. Assim sendo, os resultados da aplicação desta fórmula ao mercado financeiro não são favoráveis. Porém, este modelo foi só o início. Inspirados neste artigo, existiram muitos avanços na área financeira e surgiram novos modelos que suavizaram as restrições que o modelo de Black-Scholes possui. Alguns dos modelos mais conhecidos são o modelo de Hull e White (1987), modelo de Heston (1993), modelo de Merton (1976) e o modelo de Corrado e Su (1996).

Embora sejam uma melhoria em termos de precisão, o custo computacional aumenta exponencialmente, o que não os torna uma solução muito mais fiável. Além que continuam a não ser modelos perfeitos aos olhos do mercado, contendo ainda restrições como ausência

de arbitragem e suposições de um mercado financeiro ideal. Portanto, para avaliarmos de forma eficaz os derivados financeiros, iniciou-se outra área de pesquisa – os *machine learning algorithms*. Esta área vê as opções como funções cujos valores de input são os termos do contrato e os valores de output são o preço da opção. A aplicação de *machine learning algorithms* para a avaliação de opções oferece várias vantagens como:

- Eficiência: conseguem analisar grandes volumes de dados de forma rápida e eficiente; conseguem processar uma quantidade enorme de informações e extrair padrões relevantes em um curto período de tempo;
- Complexidade: capazes de lidar com conjuntos de dados complexos e capturar relações entre as variáveis. Eles podem identificar padrões não lineares e extrair informações valiosas que podem não ser evidentes;
- Precisão: têm a capacidade de aprender a partir dos dados, treinados em conjuntos de dados históricos por exemplo, e utilizar esse conhecimento para fazer previsões;
- Adaptação a mudanças: adaptam-se a novos dados e situações de forma flexível; conseguem atualizar os modelos e ajustar as previsões com base em novas informações;
- Automatização: é possível automatizar o processo desde que o algoritmo esteja configurado corretamente pois consegue executar a análise de forma autónoma, sem a necessidade de intervenção humana constante.

Estas características tornam esses algoritmos ferramentas valiosas para o auxílio na compra de opções. Como exemplos de métodos de *machine learning* existem o *Support Vector Regression*, o *Random Forest*, o *XGBoost*, o *LightGBM* e o *Neural Network*.

1.1. Support Vector Regression

Para entendermos o método *Support Vector Regression*, temos de perceber três conceitos base: problema de classificação, problema de regressão e o método *Support Vector Machine*.

Um problema de classificação é uma tarefa de *machine learning* onde o objetivo é prever a categoria ou classe de uma amostra baseada em características conhecidas dela. Por exemplo, dada uma imagem de uma fruta, a classificação pode prever se é uma maça ou uma laranja com base nas cores, formas e outras características da imagem. A classificação é um tipo de *supervised learning* onde o algoritmo é treinado com um conjunto de dados previamente rotulados para prever a categoria correta de novas amostras.

Um problema de regressão é uma tarefa de *machine learning* onde o objetivo é prever um valor contínuo (ou uma série de valores contínuos) a partir de um conjunto de dados. É diferente de um problema de classificação, onde o objetivo é prever uma categoria discreta. Alguns exemplos de problemas de regressão incluem prever o preço de uma casa com base em características como tamanho, localização e número de quartos ou prever o consumo de energia com base na temperatura e hora do dia.

O método *Support Vector Machine* (SVM) é um algoritmo de *supervised learning* usado para resolver problemas de classificação e regressão. O objetivo é encontrar uma linha ou um hiperplano que melhor separa os dados em duas ou mais classes. A linha ou hiperplano são escolhidos de tal forma que a distância entre eles e os pontos mais próximos de cada classe (chamados de vetores de suporte) seja o maior possível. O SVM é conhecido pela sua capacidade de lidar bem com conjuntos de dados complexos e não lineares, além de ser eficiente em termos de recursos computacionais.

O método *Support Vector Regression*(SVR) é uma extensão do algoritmo de *supervised learning Support Vector Machine* para lidar com problemas de regressão. O objetivo é encontrar um hiperplano que melhor se ajuste aos dados, ao invés de separá-los em classes como em problemas de classificação. O SVR é útil para prever valores contínuos e é especialmente útil quando os dados possuem ruídos ou *outliers*.

Seja $(x_1, y_1), ..., (x_t, y_t) \subset H \times \mathbb{R}$ um conjunto de dados de treino, onde H representa um espaço multidimensional determinado pelos parâmetros já conhecidos. O objetivo é encontrar a função $f(x)$ que tem no máximo um desvio ϵ de y_i para todo o conjunto de treino e uma inclinação tão pequena quanto possível.

Para esse efeito, a SVR formula um problema de otimização que tenta encontrar o valor de ϵ que minimiza a distância entre os valores previstos $f(x)$ e os desejados y_i.

Por exemplo, se admitirmos que H é um espaço unidimensional, a função f será da forma:

$$y = f(x) = \sum_{j=1}^{M} w_j x_j + b, \tag{1.1}$$

com $y, b \in \mathbb{R}$, $x, w \in \mathbb{R}^M$ e M o grau do polinómio usado para aproximar a função. Então, a função objetivo é dada por:

$$\min \frac{1}{2}\|w\|^2, \tag{1.2}$$

sujeita a

$$|y_i - \langle w, x_i \rangle - b| \leq \epsilon, \tag{1.3}$$

sendo ‖w‖ a magnitude do vetor normal à superfície formada pelo conjunto de treino.

A restrição (função de perda) tem como objetivo minimizar o erro entre o valor previsto da função e um determinado ponto que já conhecemos. Este método adota uma função de perda insensível a ϵ, penalizando previsões que estão mais longe do que ϵ do valor desejado.

Assim sendo, o intuito é ter o maior número de instâncias dentro da região que é insensível ao erro ϵ, no entanto de forma a que esse seja o menor erro possível. Contudo, os dados não são perfeitos e existem valores atípicos, os chamados de *outliers*. De modo a ter isso em conta, são adicionadas as variáveis ξ, ξ^* à função objetivo e que determinam a tolerância permitida aos pontos que não pertencem à região insensível a ϵ. Assim, a função objetivo fica:

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i + \xi_i^*, \tag{1.4}$$

sujeita a

$$\begin{cases} y_i - \langle w, x_i \rangle - b \leq \epsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases}, \tag{1.5}$$

sendo C uma constante.

Resolvendo este problema pelo método dos Multiplicadores de Lagrange, temos que o Langrangiano é dado por:

$$\begin{aligned} \mathcal{L} = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} b(\xi_i + \xi_i^*) - \sum_{i=1}^{N}(\eta_i\xi_i + \eta_i^*\xi_i^*) - \\ - \sum_{i=1}^{N} \alpha_i(\epsilon + \xi_i - y_i + \langle w, x_i \rangle + b) - \sum_{i=1}^{N} \alpha_i^*(\epsilon + \xi_i^* - y_i + \langle w, x_i \rangle + b), \end{aligned} \tag{1.6}$$

com $\alpha_i, \alpha_i^*, \eta_i, \eta_i^* \leq 0$. Então, as suas derivadas parciais são:

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \Leftrightarrow \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0, \tag{1.7}$$

$$\frac{\partial \mathcal{L}}{\partial w} = 0 \Leftrightarrow w - \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)x_i = 0 \Leftrightarrow w = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)x_i, \tag{1.8}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = 0 \Leftrightarrow C - \alpha_i - \eta_i = 0 \Leftrightarrow C = \alpha_i + \eta_i \tag{1.9}$$

e

$$\frac{\partial \mathcal{L}}{\partial \xi_i^*} = 0 \Leftrightarrow C - \alpha_i^* - \eta_i^* = 0 \Leftrightarrow C = \alpha_i^* + \eta_i^*. \tag{1.10}$$

Assim, substituindo no Langragiano as equações 1.7, 1.8, 1.9 e 1.10 adequadamente, resulta o problema de otimização dual:

$$\max -\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle x_i, x_j\rangle - \epsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i^*), \tag{1.11}$$

sujeito a

$$\begin{cases} \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \\ \alpha_i, \alpha_i^* \in [0, C] \end{cases}. \tag{1.12}$$

Para funções $f(x)$ não lineares, os dados são manipulados e transformados do espaço de características para o espaço de Kernel de forma a alcançar uma maior precisão. Admitindo que $k(.)$ é a função de Kernel e que $k(x, x') = \langle \Phi(x), \Phi(x')\rangle$, temos que o problema referido a cima pode ser reescrito como:

$$\max -\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)k(x_i, x_j) - \epsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i^*), \tag{1.13}$$

sujeito a

$$\begin{cases} \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \\ \alpha_i, \alpha_i^* \in [0, C] \end{cases}, \tag{1.14}$$

com

$$w = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\Phi(x_i) \tag{1.15}$$

e

$$f(x) = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)k(x_i, x) + b. \tag{1.16}$$

Aplicando de novo as equações 1.7, 1.8, 1.9 e 1.10 ao novo problema de otimização usando a função *Radial Basis Function* como a função kernel, temos que o langrangiano deste problema é dado por:

$$\mathcal{L} = \frac{1}{2}(\|w\| + b) + \frac{C}{2}\sum_{i=1}^{l} p_\epsilon^2(k(x_i, x_j)w + b - y_i, a), \tag{1.17}$$

onde

$$p_\epsilon^2 = p(x - \epsilon, a)^2 + p(-x - \epsilon, a)^2, \tag{1.18}$$

$$p(x, a) = x + \frac{1}{a}\log(1 + \exp(-ax)) \tag{1.19}$$

com $a > 0$ e

$$k(x_i, x) = \exp(-\gamma\|x_i - x\|^2). \tag{1.20}$$

1.2. Random Forest

Para percebermos este método, necessitamos de entender o conceito de árvore de decisão: uma árvore de decisão é uma estrutura de dados que modela as decisões e as suas consequências sobre uma série de escolhas. Esta estrutura é composta por um conjunto de nós, onde cada nó representa uma decisão, e um conjunto de arestas que coneta os nós, representando as possíveis consequências. Na análise de dados, uma árvore de decisão é um modelo que se utiliza para prever o valor de uma variável com base em várias variáveis previamente conhecidas.

Random Forest é um algoritmo de *supervised machine learning* muito utilizado devido à sua precisão, simplicidade e flexibilidade. A designação "Forest" deve-se à formação de uma floresta de árvores de decisão para realizar a classificação ou regressão de um conjunto de dados. Os dados do conjunto de treino são divididos em grupos e cria-se, de forma aleatória, uma árvore de decisão para cada um dos grupos. Essas árvores de decisão são independentes e providenciam uma previsão individual para cada subconjunto de dados. O método combina todas essas previsões num resultado que, no caso de ser um problema de classificação, encontra-se escolhendo a classe mais comum e no caso de ser um problema de regressão, encontra-se realizando a média de todas as previsões. Este método é considerado a técnica mais poderosa e eficiente de *machine learning* pois é capaz de suportar conjuntos de dados numerosos e complexos, com alta dimensionalidade e variabilidade.

Este método é complementado por duas componentes: o *Bagging* e o Critério de Divisão de Árvores de Classificação e Regressão (CART). O *Bagging* é um poderoso método que foi proposto por Leo Breiman em 1994 para evitar o *overfitting*. Este método é a aplicação do procedimento de *bootstrap* que se trata de um método de reamostragem e baseia-se na criação de numerosas amostras aleatórias do conjunto de dados original. Como as árvores de decisão têm uma variância muito elevada, utiliza-se o método *Bagging* que reduz a variância dos modelos. Cada modelo é utilizado em cada amostra de dados que sofreram *bootstrap* e o resultado médio desses modelos é a previsão pretendida. Em relação ao CART, o conjunto de dados é dividido em subconjuntos cada vez menores com base nas características dos dados que já conhecemos, até que os subconjuntos finais sejam tão homogéneos quanto possível em relação ao valor a ser previsto. Essa estratégia de divisão é repetida recursivamente até atingir o critério de paragem, como dimensão da árvore ou o número mínimo de amostras em cada folha da árvore.

1.3. XGBoost

Extreme Gradient Boosting, abreviado para *XGBoost*, é um algoritmo de *machine learning* que se concentra nos modelos baseados em árvores de decisão e é usado, maioritariamente, para problemas de classificação e regressão. Este algoritmo é baseado no algoritmo de *boosting* que funciona treinando sucessivamente modelos fracos em cima dos erros do modelo anterior até que um modelo forte seja construído. O *XGBoost* adiciona melhorias significativas a este processo de *boosting*, incluindo a regularização L_1 (adiciona uma penalidade à função de custo do modelo proporcional ao valor absoluto dos coeficientes das variáveis independentes) e L_2 (adiciona uma penalidade proporcional ao quadrado dos coeficientes) para evitar o *overfitting*.

Este modelo pode lidar com conjuntos de dados muito grandes e é considerado um dos algoritmos mais poderosos para problemas de *machine learning* devido à sua capacidade de lidar com dados não lineares e de alta dimensão.

Dado um conjunto $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ de valores conhecidos de x e valores correspondentes de y, com $x_i \in \mathbb{R}^m$ e $y_i \in \mathbb{R}$, um modelo de conjunto de árvores (como ilustrado na 1.3 que foi retirada no artigo [**4**]), usa K funções aditivas para prever o output:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \tag{1.21}$$

com $f_k \in \mathcal{F}$; onde $\mathcal{F} = \{f(x) = w_{q(x)}\}$ com $q : \mathbb{R}^m \to T$, $w \in \mathbb{R}^T$ é o espaço de árvores de regressão. O q representa a estrutura de cada árvore que faz a correspondência de um ponto para o índice correspondente da folha e o T é o número de folhas na árvore. Cada f_k corresponde a uma estrutura de árvore independente q e pesos de folha w. Cada árvore contém uma pontuação contínua em cada folha designada por w_i.

FIGURA 1. Conjunto de árvores de decisão



Em resumo, o modelo *XGBoost* usa várias árvores de decisão independentes para fazer previsões e combina os seus resultados para produzir uma previsão final. Cada árvore é treinada em diferentes amostras aleatórias do conjunto de dados e com diferentes conjuntos de recursos para evitar *overfitting* e melhorar a generalização do modelo.

Para um ponto já conhecido, usamos as regras de decisão nas árvores (dadas por q) para classificá-lo nas folhas e calcular a previsão final somando a pontuação nas folhas correspondentes (dadas por w). Para calcular o conjunto de funções usadas no modelo,

minimizamos a seguinte função objetivo regularizada:

$$\mathcal{L}(\phi) = \sum_{i=1}^{m} l(\hat{y}_i, y_i) + \sum_{k=1}^{K} \Omega(f_k), \tag{1.22}$$

com $\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$ onde l é uma função de perda convexa diferenciável que mede a diferença entre a previsão $\hat{y}_i$ e o alvo y_i. O segundo termo Ω penaliza a complexidade do modelo (ou seja, as funções de árvore de regressão).

Esta equação inclui funções como parâmetros e, por isso, não pode ser otimizada usando métodos tradicionais de otimização no espaço euclidiano. Em vez disso, o modelo é trabalhado de maneira aditiva. Denotemos $\hat{y_i}^{(t)}$ a previsão da i-ésima instância na t-ésima interação e precisamos de adicionar f_t para minimizar a seguinte equação objetivo:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{m} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t). \tag{1.23}$$

1.4. LightGBM

Um *framework* de *boosting* de gradientes é uma ferramenta que permite criar modelos de *machine learning* usando a técnica de *boosting* de gradientes. Esta técnica consiste na combinação de vários modelos fracos em um modelo forte e preciso.

O método *Light Gradient Boosting Machine* é um *framework* de *boosting* de gradientes desenvolvido pela *Microsoft* que é baseado em árvores de decisão. Este método combina duas técnicas novas: o *Gradient-based One Side Sampling* (GOSS) e o *Exclusive Feature Bundling* (EFB).

A técnica GOSS baseia-se no pressuposto que diferentes instâncias de dados têm papéis variados na forma como se recolhe informação. Assim, as instâncias com maiores gradientes irão contribuir mais para a recolha de informação do que as instâncias com menores gradientes. Portanto, esta técnica mantém as instâncias de maior gradiente e descarta as restantes para manter a precisão de estimativa da recolha de informação.

A técnica EFB baseia-se na questão que os dados de alta dimensão são normalmente muito dispersos e que isso oferece a possibilidade de delinear uma abordagem, praticamente sem perdas, para reduzir o número de características desses dados. Portanto, num espaço de características disperso, muitas características são mutuamente exclusivas, ou seja, nunca assumem valores diferentes de zero simultaneamente. Assim, essas características exclusivas podem ser agrupadas em uma única característica. Logo, esta técnica reduz o número de características diminuindo também, por consequência, a complexidade da construção do histograma.

Estas técnicas conjugam-se e tornam o modelo mais eficiente e com um menor custo computacional.

1.5. Neural Networks

Neural networks são um subconjunto de *machine learning* que processa os dados de forma semelhante ao cérebro humano, imitando a forma como os neurónios biológicos transmitem a mensagem. Ou seja, nas redes neurais o intuito é organizar a estrutura dos dados como se se tratassem de neurónios num cérebro e, tal como o cérebro humano, aprender e tomar decisões baseadas nessas aprendizagens. As redes neurais são compostas por nós. Cada nó (neurónio artificial) está conectado a outro nó e tem um peso associado. Esses pesos são obtidos através de um processo de aprendizagem a partir de um conjunto de dados de treino.

Existem vários tipos de redes neurais, no entanto o mais conhecido é o *Multilayer Perpectron* (MLP). Este modelo é composto por várias camadas de nós (também designados por *perceptron*), contendo uma camada do *input*, onde são introduzidas as variáveis independentes, uma ou mais camadas ocultas e uma camada do *output* onde se encontram as variáveis a serem estimadas.

A funcionalidade de um nó singular é simples. A aplicação deste conceito ao problema de decisão de ir ao cinema é um exemplo da função de um nó. A decisão de ir ou não é a variável a ser estimada e as variáveis independentes são os valores binários que respondem aos fatores que influenciam a decisão. Ou seja, a questão "Gosto de ver filmes?"em que a resposta "Sim"dá o valor 1 a essa variável independente é um exemplo de um fator que influencia o resultado. Dependendo da importância desse fator, teremos que essa variável independente tem um maior ou menor peso. O valor previsto é calculado como a soma dos produtos das variáveis independentes com o respetivo peso adicionando um termo de viés.

Para um problema mais complexo e não-linear, temos que cada nó soma todos os sinais enviados a ele, adiciona um termo de polarização e realiza uma transformação não linear. A função de ativação (que determina quando existe transferência de sinais entre os nós) é uma função monótona e crescente. A função do erro é dada pela comparação do valor dos nós do *output* e o valor real dos dados. O valor dos pesos é ajustado de forma que a função do custo seja minimizada até o modelo estimar os dados do *output* com uma precisão aceitável. Este algoritmo é denominado de retro-propagação.

2

Dados e Metodologia

Para uma avaliação empírica de opções através de métodos de *machine learning*, utilizamos o preço das opções de compra de estilo europeu SPX7, que representam as opções mensais do índice S&P500, entre 03-01-2022 e 31-05-2022. Foram analisados 249 575 preços diários. Estes dados foram providenciados pelo Professor João Pedro Ruas e pelo Refinitiv.

Foram utilizadas opções de compra de estilo europeu com as seguintes características:

- Maturidade inferior a um ano;
  Estas opções têm uma maior liquidez e, por terem um prazo mais curto, existem mais compradores e vendedores ativos no mercado, o que tende a reduzir a diferença entre os preços de compra e de venda das opções (também conhecido por *spread*). Com uma maior liquidez e um *spread* ajustado, a credibilidade nos preços das opções aumenta.
- *Moneyness* inferior a 1.7;
  Moneyness representa a relação entre o preço de exercício K (*strike price*) e o preço do ativo subjacente S (*underlying asset price*) e é calculado por S/K. O valor limite 1.7 é utilizado pelos autores do artigo [**1**].
- *Premium* superior a 1. As opções com prémios mais elevados são mais representativas das expectativas e estratégias dos investidores que, normalmente, têm um maior interesse por opções que têm prémios significativos. Para além disso, este filtro exclui as opções com prémios muito baixos que não são relevantes para esta análise por serem mais propensas a flutuações de preço devido à menor liquidez e interesse dos *traders*. Esta diferença de comportamento em comparação com as restantes opções poderia afetar a interpretação dos resultados.

O conjunto de dados foi dividido em grupos de acordo com a maturidade e a *moneyness* de cada contrato.

Relativamente à maturidade, a divisão realizou-se de acordo com o seguinte critério:

- Contratos a curto prazo (*short term*): contratos com maturidade inferior a 3 meses;
- Contratos a médio prazo (*medium term*): contratos com maturidade entre 3 e 9 meses;

- Contratos a longo prazo (*long term*): contratos com maturidade superior a 9 meses.

Relativamente à *moneyness*, o critério aplicado foi:

- Contratos *Deep Out of The Money* (DOTM): contratos com a *moneyness* inferior a 0.8;
- Contratos *Out of The Money* (OTM): contratos com a *moneyness* entre 0.8 e 0.96;
- Contratos *At The Money* (ATM): contratos com a *moneyness* entre 0.96 e 1.04;
- Contratos *In The Money* (ITM): contratos com a *moneyness* entre 1.04 e 1.02;
- Contratos *Deep In The Money* (DITM): contratos com a *moneyness* superior a 1.02.

A avaliação dos preços das opções foi calculada com base no erro da raiz quadrática média (*root mean square error* – RMSE) para cada um dos grupos apresentados acima. Na aplicação dos métodos de *machine learning*, dividimos aleatoriamente cada um dos grupos num conjunto de treino (80% dos dados) e num conjunto de teste (20% dos dados) para a implementação do método.

O erro da raiz quadrática média é calculado através da fórmula:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=0}^{N}(x_i - \hat{x}_i)^2}{N}}, \tag{2.1}$$

assumindo que N é o número de dados da análise, x_i o valor dos dados observados e $\hat{x}_i$ o valor dos dados previstos em cada um dos métodos.

Os métodos de *machine learning* utilizados neste estudo foram o *Support Vector Regression*, o *Random Forest*, o *XGBoost*, o *LightGBM* e o *Neural Network*. Como referência, também foram utilizados o modelo Black-Scholes-Merton (1973) e o modelo de Corrado e Su (1996).

Os dados assumidos na aplicação destes métodos e modelos paramétricos foram filtrados de forma a todos os contratos admitam volatilidade histórica dos últimos 60, 30 e 10 dias. Ou seja, foram ignorados os dados cuja data de início esteja incluída nos primeiros 61 dias do intervalo de tempo selecionado para este estudo.

2.1. Implementação do modelo de Black-Scholes-Merton

Para calcular o preço das opções de compra de estilo europeu, a partir do modelo de Black-Scholes-Merton (proveniente dos artigos [**2**] e [**3**]), utilizam-se as seguintes fórmulas:

$$C_{BSM} = Se^{-q\tau}N(d_1) - Ke^{-r\tau}N(d_2) \tag{2.2}$$

$$d_1 = \frac{\log\frac{S}{K} + (r - q + \frac{1}{2}\sigma^2)\tau}{\sigma\sqrt{\tau}} \tag{2.3}$$

$$d_2 = d_1 - \sigma\sqrt{\tau}, \tag{2.4}$$

assumindo que S é o preço do ativo subjacente, K é o preço de exercício, r é a taxa de juro anual, q é a *dividend yield*, τ é o tempo para a maturidade e σ é a volatilidade. A função $N(.)$ representa a função de distribuição cumulativa normal *standard*.

A metodologia para a avaliação dos preços dos modelos paramétricos, como é o caso do modelo de Black-Scholes-Merton, baseia-se em determinar os parâmetros implícitos por meio de Mínimos Quadrados Não-Lineares. O parâmetro implícito deste modelo é a volatilidade. Assim, calculamos a volatilidade implícita de acordo com a fórmula:

$$\min_{\sigma_{imp}} \sum_{j=1}^{N} [C_{OBS} - C_{BSM,j}(\sigma_{imp})]^2, \tag{2.5}$$

assumindo que $C_{OBS,j}$ é o preço observado no mercado, $C_{BSM,j}(\sigma_{imp})$ é o preço aplicando o modelo de Black-Scholes-Merton que depende da volatilidade implícita e o N é o número total de contratos.

A volatilidade implícita foi calculada para conjuntos de contratos com a mesma data de início e a mesma maturidade. A metodologia utilizada pode ser vista, com mais detalhe, na secção A.3 e na secção A.5.

2.2. Implementação do modelo de Corrado e Su

Para calcular o preço das opções de compra de estilo europeu, a partir do modelo de Corrado e Su, com base no artigo [**5**], utilizam-se as seguintes fórmulas:

$$C_{CS} = C_{BSM} + \mu_3 Q_3 + (\mu_4 - 3)Q_4 \tag{2.6}$$

$$Q_3 = \frac{1}{3!} S\sigma\sqrt{\tau}[(2\sigma\sqrt{\tau} - d)N(d) - \sigma^2\tau N(d)] \tag{2.7}$$

$$Q_4 = \frac{1}{4!} S\sigma\sqrt{\tau}[(d^2 - 1 - 3\sigma\sqrt{\tau}(d - \sigma\sqrt{\tau}))N(d) + \sigma^3\tau^{\frac{3}{2}}N(d)] \tag{2.8}$$

$$d = \frac{\log\frac{S}{K} + (r - q + \frac{1}{2}\sigma^2)}{\sigma\sqrt{\tau}}, \tag{2.9}$$

assumindo que C_{BSM} é o preço da opção calculado pelo modelo de Black-Scholes-Merton, μ_3 é a *skewness*, μ_4 é a *kurtosis*, S é o preço do ativo subjacente, K é o preço de exercício, r é a taxa de juro anual, q é a *dividend yield*, τ é o tempo para a maturidade e σ é a volatilidade. A função $N(.)$ representa a função de distribuição cumulativa normal *standard*.

À semelhança do modelo de Black-Scholes, a metodologia para a avaliação dos preços baseia-se em determinar os parâmetros implícitos por meio de Mínimos Quadrados Não-Lineares. Os parâmetros implícitos deste modelo são a volatilidade, *skewness* e *kurtosis* e foram calculados através da fórmula:

$$\min_{\sigma_{imp},\mu_{3imp},\mu_{4imp}} \sum_{j=1}^{N} [C_{OBS,j} - (C_{BSM,j}(\sigma_{imp}) + \mu_{3imp}Q_3 + (\mu_{4imp} - 3)Q_4]^2, \tag{2.10}$$

assumindo que $C_{OBS,j}$ é o preço observado no mercado, $C_{BSM,j}(\sigma_{imp})$ é o valor do modelo de Black-Scholes-Merton que depende da volatilidade implícita e N é o número total de contratos.

Analogamente, estes parâmetros implícitos foram calculados para conjuntos de contratos com a mesma data de início e a mesma maturidade. A metodologia utilizada pode ser vista, com mais detalhe, na secção A.4 e na secção A.6.

2.3. Volatilidade Histórica

De forma a evidenciar a diferença entre utilizar parâmetros implícitos e parâmetros históricos, calculámos a volatilidade histórica dos últimos 60, 30 e 10 dias para cada um dos contratos.

A volatilidade histórica é calculada a partir dos retornos dos preços do ativo subjacente. Assim, o retorno R_i para cada uma das datas de início i é:

$$R_i = \log \frac{S_i}{S_{i-1}}. \tag{2.11}$$

Então, a volatilidade histórica dos últimos n dias de uma opção com data de início t é

$$\sigma_n = \sqrt{\frac{\sum_{i=1}^{t+n}(R_i - \bar{R}_t)^2}{n-1}} \tag{2.12}$$

com $\bar{R}_t$ a média dos retornos que é calculado por:

$$\bar{R}_t = \frac{1}{n}\sum_{i=t-n}^{t} R_i, \text{ com } t > n. \tag{2.13}$$

Para o caso em que $t < n$ não é possível calcular a volatilidade histórica dos últimos n anos.

A implementação desta metodologia pode ser vista na secção A.2.

2.4. Implementação do método Support Vector Regression

Os parâmetros do método *Support Vector Regression* são a função de *kernel*, o nível de tolerância para o critério de paragem (*tol*), a constante de penalidade C e o valor de ϵ que determina a margem de erro nos dados do conjunto de treino.

A função de *kernel* é utilizada para transformar as características dos dados do *input* em um espaço de alta dimensão onde o hiperplano que ajusta os dados pode ser construído. A função de *kernel* utilizada foi a *Radial Basis function* (RBF) uma vez que é a função mais eficaz para problemas não lineares. Esta função é definida como $K(x, y) = \exp(-\gamma\|x_i - x\|^2)$, com $\gamma = (\text{quantidade de características dos dados})^{-1}$.

A constante de penalidade C constata o *trade-off* entre maximizar a margem entre os dados e o hiperplano de regressão e minimizar o erro. O valor assumido para este parâmetro é 1.

Para os parâmetros *tol* e ϵ são assumidos os valores 0.001 e 0.1, respetivamente. Os valores admitidos nos parâmetro deste método fundamentam-se na escolha do autor do artigo [**1**].

A implementação deste método tem por base a função apresentada em [**6**] e pode ser vista na secção A.7.

2.5. Implementação do método Random Forest

Os parâmetros analisados no método *Random Forest* foram o número de árvores de decisão e a profundidade máxima de cada árvore.

Quanto maior for o número de árvores de decisão, maior será o desempenho do modelo, tornando-o mais preciso. No entanto, se o número de árvores de decisão for muito elevado, a eficiência do método diminui drasticamente. É necessário encontrar um meio termo.

Relativamente à profundidade de cada árvore, quanto menos níveis a árvore admitir, menos complexas e mais simples serão. Porém, árvores de decisão profundas podem ser muito específicas para o conjunto de dados de treino e, consequentemente, pode levar a *overfitting*, para além de necessitarem de mais recursos computacionais.

Os valores assumidos para estes parâmetros foram 100 e 20, respetivamente e fundamentam-se na escolha do autor do artigo [**1**].

A implementação deste método tem por base a função apresentada em [**7**] e pode ser vista na secção A.8.

2.6. Implementação do método XGBoost e do método LightGBM

Os parâmetros dos métodos *XGBoost* e *LightGBM* são a taxa de aprendizagem e a profundidade de cada árvore de decisão.

A taxa de aprendizagem é o valor que se multiplica às atualizações dos gradientes durante o processo. Uma maior taxa de aprendizagem permite que o método aprenda mais rapidamente, no entanto é mais sensível às oscilações dos dados.

Relativamente à profundidade das árvores de decisão, árvores mais profundas permitem capturar padrões mais complexos nos dados, no entanto podem levar ao *overfitting* do conjunto dos dados de treino. Normalmente, em conjuntos de dados menores, usa-se árvores mais profundas para haver um maior ajuste.

Para o parâmetro da profundidade de cada árvore de decisão assumimos 20 níveis de profundidade, tendo em conta o artigo [**1**].

Para o parâmetro da taxa de aprendizagem, os métodos apresentam um menor erro quando aplicável a taxa de 0.1 em vez de 0.3, como apresentado no artigo [**1**]. Portanto, este parâmetro vai assumir o valor de 0.1.

A implementação do método de *XGBoost* tem por base a função apresentada em [**8**] e pode ser vista na secção A.9. A implementação do método *LightGBM* tem por base a função apresentada em [**9**] e pode ser vista na secção A.10.

2.7. Implementação do método Neural Networks

Para o método *Neural Networks* não existe uma função integrada, então é necessário construir uma rede neural. Com esse intuito, adicionam-se camadas para se criar uma rede e cada camada é constituída por "nós". Foram construídas três camadas: uma com 16 nós, uma com 8 nós e a última apenas com um nó - o *output*.

Os parâmetros para ajustar o modelo são o número de iterações e o tamanho do lote.

O número de iterações controla quantas vezes o método percorre os dados do conjunto de treino. Deve encontrar-se um valor equilibrado para se evitar o *overfitting*.

No método das redes neurais, os dados do conjunto de treino são usados aos lotes até se usarem todos os dados do conjunto de treino. O tamanho do lote refere-se à quantidade de dados em cada um dos lotes. A escolha deste parâmetro afeta positivamente a velocidade do método caso sejam lotes maiores, no entanto lotes menores admitem uma convergência mais suave dos dados.

Para este método admitimos que o número de iterações é 100 e que o tamanho do lote é 32. A implementação pode ser vista na secção A.11.

3

Resultados

Neste capítulo são apresentados os resultados empíricos dos métodos de *machine learning* e dos modelos Black-Scholes-Merton e Corrado e Su.

De forma a concluirmos o melhor espaço de características, ou seja, o melhor conjunto de variáveis utilizadas para o *input*, foi feito um estudo com 5 *inputs* distintos. Para cada um dos *inputs*, foi calculado o erro RMSE de cada um dos métodos de *machine learning* e dos outros dois modelos, sendo que o modelo de Black-Scholes-Merton foi calculado admitindo a volatilidade histórica dos últimos 60 dias e a volatilidade implícita. Como esperado, no modelo de Black-Scholes-Merton, o erro é menor quando se assume a volatilidade implícita. Para o modelo de Corrado e Su, todos os parâmetros utilizados são os implícitos calculados na secção 2.2. Os parâmetros em comum para todos os modelos são o S que representa o preço do ativo subjacente, o K que é o *strike price*, o τ que é o tempo expresso em anos até à data de vencimento do contrato, o r que é a taxa de risco sem juros e o σ_n que é a volatilidade dos últimos n anos. Os resultados destes vários *inputs* podem ser vistos na tabela 3.

Tabela 1. Avaliação do erro RMSE do preço das opções para cinco grupos de *inputs* diferentes

	G1	G2	G3	G4	G5
Input	S/K, τ	S, K, τ	S, K, τ, r	S, K, τ, r, σ_{60}	S, K, τ, r, σ_{60}, σ_{30}, σ_{10}
Output	C/K	C	C	C	C
BS_{imp}	30.194	30.194	30.194	30.194	30.194
$BS_{\sigma_{60}}$	105.369	105.369	105.369	105.369	105.369
CS	25.363	25.363	25.363	25.363	25.363
SVR	0.120	82.837	82.837	82.837	82.837
RF	0.004	3.704	3.850	3.802	3.780
XGB	0.003	1.868	1.945	1.968	1.965
LGBM	0.004	7.905	7.797	7.972	7.827
NN	0.004	7.519	6.979	8.332	7.950

O grupo 1 de *inputs* é baseado na proposta de [**10**]. Este artigo defende que a função de avaliação do preço de opções é homogénea de grau um em S e K, ou seja, $f(S, K, \dots) = f(S/K, 1, \dots)$. Assim, os *inputs* neste conjunto são S/K e τ. Neste caso, todos os métodos de *machine learning* superaram os modelos de Black-Scholes-Merton e Corrado e Su.

O grupo 2 de *inputs* utiliza os mesmos parâmetros que o conjunto 1, no entanto sem assumir que a função de avaliação do preço de opções é homogénea. Assim, os *inputs* neste conjunto são S, K e τ. O único algoritmo que não superou os dois modelos foi o SVR. Os algoritmos de árvores de decisão RF e XGB apresentaram, de longe, o melhor desempenho.

O grupo 3 de *inputs* utiliza os parâmetros S, K, τ e r. O grupo 4 de *inputs* utiliza os parâmetros S, K, τ, r e σ_{60}. O grupo 5 de *inputs* utiliza os parâmetros S, K, τ, r, σ_{60}, σ_{30} e σ_{10}. À semelhança do grupo 2, nestes grupos destacam-se os métodos RF e XGB, sendo que o método que apresenta um menor erro é o XGB.

Os melhores resultados foram obtidos utilizado o grupo G1 de *inputs*, no entanto estes resultados são muito discrepantes dos erros dos outros grupos. Assim, para as próximas análises serão obtidas com base no grupo G3 que apresenta os melhores resultados, não tendo em conta o grupo G1.

Para um estudo mais detalhado da avaliação do preço das opções, é apresentada a tabela 3 que é dividida consoante o *moneyness* (*Deep In The Money*, *In The Money*, *At The Money*, *Out of The Money* e *Deep Out of The Money*) e a maturidade (*short term*, *medium term* ou *long term*) de cada contrato.

Tabela 2. Avaliação do erro RMSE do preço das opções para o grupo 3 de *inputs* dividido em *moneyness* e maturidade

	Métodos	Tudo	DITM	ITM	ATM	OTM	DOTM
Tudo	BS_{imp}	30.194	16.161	31.588	13.201	35.160	38.336
	BS_{60}	105.369	5.593	74.836	187.431	120.447	56.367
	CS	25.363	17.9761	36.749	13.990	21.233	18.807
	SVR	82.837	5.365	50.524	89.521	75.944	81.829
	RF	3.854	0.453	2.332	4.162	4.706	6.317
	XGB	1.945	0.335	1.243	2.422	2.549	2.904
	LGBM	7.797	0.375	2.560	5.031	7.394	7.166
	NN	7.237	0.507	3.192	9.939	14.304	5.820
Short term	BS_{imp}	14.219	0.863	13.930	10.464	20.001	11.020
	BS_{60}	52.526	0.162	21.119	104.086	52.022	12.349
	CS	12.155	0.923	15.127	11.083	13.703	233.159
	SVR	208.189	0.151	11.277	34.729	36.040	5.097
	RF	2.855	0.111	0.767	2.359	3.306	3.520
	XGB	1.751	0.115	0.558	1.850	2.194	5.795
	LGBM	7.115	0.106	0.956	3.690	5.456	6.414
	NN	6.052	0.113	2.437	10.539	9.943	4.168
Medium term	BS_{imp}	33.504	13.577	36.633	14.324	40.426	39.767
	BS_{60}	116.335	3.010	78.464	213.872	136.858	54.159
	CS	28.039	14.932	42.044	16.304	24.732	19.622
	SVR	182.712	2.220	26.695	43.344	46.538	177.128
	RF	4.462	0.281	2.443	4.761	5.438	6.767
	XGB	2.181	0.207	1.397	2.878	3.163	3.665
	LGBM	5.995	0.225	2.574	5.154	6.378	6.279
	NN	4.833	0.584	3.708	9.878	9.529	9.256
Long term	BS_{imp}	47.905	28.007	50.606	18.365	55.825	60.67
	BS_{60}	165.877	10.953	143.186	16.003	215.082	96.933
	CS	40.275	31.315	60.964	15.494	30.173	29.463
	SVR	407.738	3.530	46.279	47.037	86.997	228.156
	RF	5.386	0.650	4.420	7.800	7.999	7.780
	XGB	2.573	0.521	1.972	4.397	4.255	4.638
	LGBM	4.828	0.520	3.133	6.281	6.092	7.413
	NN	640.160	0.892	5.585	61.632	66.675	33.985

À semelhança das conclusões do estudo apresentado na tabela 3, na tabela 3, o método que apresenta, na maioria dos casos, um menor erro, independentemente da maturidade ou *moneyness*, é o XGB. No entanto, é observável que os erros dos métodos de machine learning são muito inferiores aos do modelo de Black-Scholes-Merton e do Corrado e Su. Como esperado, o modelo com a pior performance é o modelo de Black-Scholes-Merton com a volatilidade histórica.

Para uma avaliação mais clara, a tabela 3 apresenta, dependendo da maturidade e *moneyness*, o método que apresenta um erro menor.

Tabela 3. Método que apresenta o menor erro RMSE dos dados da tabela 3

	Tudo	DITM	ITM	ATM	OTM	DOTM
Tudo	XGB	XGB	XGB	XGB	XGB	XGB
Short term	XGB	LGBM	XGB	XGB	XGB	RF
Medium term	XGB	XGB	XGB	XGB	XGB	XGB
Long term	XGB	LGBM	XGB	XGB	XGB	XGB

4

Conclusões

Nesta dissertação foram estudadas opções de compra mensais do estilo europeu do índice S&P500 com o intuito de avaliar os seus preços. Para isso, foram apresentados dois modelos teóricos - o modelo Black-Scholes-Merton e o modelo Corrado e Su - e seis métodos de *machine learning* - o *Support Vector Regression*, o *Random Forest*, o *Extreme Grandient Boosting*, o *Light Gradient Boosting Machine* e o *Neural Networks*.

O erro da aplicação dos modelos e métodos com o valor real de mercado das opções foi calculado através do *root mean square error*. A performance dos métodos de *machine learning* superaram os resultados dos modelos teóricos, em todos os conjuntos de dados analisados. Entre os métodos de *machine learning* destacaram-se os métodos de árvores de decisão: o XGBoost e o LightGBM, que também correspondem aos algoritmos mais recentes. Os dois modelos utilizam a mesma ferramenta de base: combinar modelos mais fracos e assim criar um modelo forte e preciso. A pouca diferença nos resultados pode dever-se a esta semelhança.

Referências Bibliográficas

[1] IVASCU, Codrut-Florin. "Option pricing using Machine Learning". Expert Systems With Applications, 2021.

[2] BLACK, Fischer; SCHOLES, Myron. "The Pricing of Options and Corporate Liabilities". The Journal of Political Economy, 1973.

[3] MERTON, Robert. "Theory of Rational Option Pricing". The Bell Journal of Economics and Management Science, 1973.

[4] CHEN, Tianqi; GUESTRIN, Carlos. "XGBoost: A Scalable Tree Boosting System". KDD '16, August 13-17, 2016, San Francisco, CA, USA.

[5] CORRADO, Charles; SU, Tie. "Skewness and kurtosis in S&P 500 index returns implied by option prices". The Journal of Financial Research, Vol XIX, No. 2, Pages 175-192, 1996.

[6] SATURN CLOUD. "A Comprehensive Guide to Support Vector Regression Parameters using LibSVM in MATLAB". Saturn Cloud, 2023. Obtido de https://saturncloud.io/blog/a-comprehensive-guide-to-support-vector-regression-parameters-using-libsvm-in-matlab/epsilon-parameter

[7] SCIKIT LEARN, "sklearn.ensemble.RandomForestRegressor". Scikit Learn. Obtido de https://scikit-learn.org/sklearn.ensemble.RandomForestRegressor.html

[8] BROWNLEE, Jason. "XGBoost for Regression". Machine Learning Mystery, 2021. Obtido de https://machinelearningmastery.com/xgboost-for-regression/ https://machinelearningmastery.com/xgboost-for-regression/

[9] LIGHTGBM. "lightgbm.LGBMRegressor". LightGBM, 2023. Obtido de https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRegressor.html

[10] HUTCHINSON, James; LO, Andrew; POGGIO, Tomaso. "A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks". The Journal of Finance, 1994.

A

Códigos em Python

Neste capítulo serão apresentados os códigos em linguagem Python utilizados na dissertação.

Os códigos estão encadeados, podendo ser utilizadas variáveis ou funções de excertos de códigos de secções anteriores.

A.1. Tratamento dos Dados

```
import pandas as pd
import glob
import warnings
warnings.filterwarnings("ignore")

def comp_ficheiros (url_SPX_Opt):
    # SPX options data
    fp_SPX = url_SPX_Opt

    # Get all .csv files in fp_SPX
    files_list = glob.glob(fp_SPX + "/*.csv")

    # Selected Variables to import
    var_selected = ['root_cycle', 't', 'T', 'tau_d', 'tau', 'opt_type',
    'K', 'bid', 'mid', 'ask', 'vol', 'under_bid', 'under_mid', 'under_ask
    ', 'r', 'P', 'F','divyield', 'IV']

    # Minimum time to option expiration (in days)
    tau_d_min = 7
    tab_total = pd.DataFrame()

    for file in files_list:
        # Read SPX file
        tab = pd.read_csv(file)
        tab = tab[var_selected]
        # Apply condition of Minimum time to option expiration (in days)
        tab = tab[tab['tau_d'] >= tau_d_min]
```

```
        # Only call options
        tab_remove = tab.loc[(tab['opt_type'] == 'p')]
        tab_final = tab.drop(tab_remove.index)

        # Maturity of less than 1 year
        tab_remove = tab_final.loc[(tab_final['tau'] >= 1)]
        tab_final = tab_final.drop(tab_remove.index)

        # Moneyness (under_mid/K) less than 1.7
        tab_remove = tab_final.loc[(tab_final['under_mid']/tab_final['K'
    ] >= 1.7)]
        tab_final = tab_final.drop(tab_remove.index)

        # Premium (P) greater than 1
        tab_remove = tab_final.loc[(tab_final['P'] > 1)]
        tab_final = tab_final.drop(tab_remove.index)

        #Just Monthly options
        tab_remove = tab_final.loc[(tab_final['root_cycle'] != 'SPX7')]
        tab_final = tab_final.drop(tab_remove.index)
        tab_total = pd.concat([tab_total,tab_final])


    index = []
    for i in range(0,len(tab_total['t'])):
        index.append(i)
    tab_total ['index'] = index
    tab_total = tab_total.set_index('index')

    return tab_total
```

A.2. Volatilidade Histórica

```
import pandas as pd
import math

def add_vol_hist (tab_total):
    aux = pd.DataFrame()
    aux ['t'],aux['under_mid'] = tab_total['t'], tab_total['under_mid']
    aux.drop_duplicates(inplace = True)
    aux = aux.set_index('t')

    retorno = ['NaN']
    for numero in range(1, len(aux['under_mid'])):
```

```
        retorno.append(math.log(aux['under_mid'][numero]/aux['under_mid'
    ][numero-1]))
    aux['Retorno'] = retorno

    def volatilidade_historica(n):
        vol=0
        coluna=[]
        for i in range(0,n+1):
            coluna.append('NaN')
        for i in range (n+1, len(aux['under_mid'])):
            media = aux['Retorno'][i-n:i-1].mean()
            for numero in range(0,n):
                vol = vol + (aux['Retorno'][i-numero] - media)**2
            vol = vol/(n-1)
            vol = vol ** (1/2)
            coluna.append(vol)
            vol = 0
        return coluna
    aux['HV60'] = volatilidade_historica(60)
    aux['HV30'] = volatilidade_historica(30)
    aux['HV10'] = volatilidade_historica(10)

    dic60 = {}
    dic30 = {}
    dic10 = {}
    for a in range(len(aux['under_mid'])):
        dic60[aux.index[a]] = aux['HV60'][a]
        dic30[aux.index[a]] = aux['HV30'][a]
        dic10[aux.index[a]] = aux['HV10'][a]
    tab_total ['HV60'] = 0
    tab_total ['HV30'] = 0
    tab_total ['HV10'] = 0

    for a in range(len(tab_total['under_mid'])):
        tab_total ['HV60'] [a] = dic60[tab_total['t'][a]]
        tab_total ['HV30'] [a] = dic30[tab_total['t'][a]]
        tab_total ['HV10'] [a] = dic10[tab_total['t'][a]]
```

A.3. Volatilidade Implícita - Black-Scholes-Merton

```
import pandas as pd
from scipy.stats import norm
from scipy.optimize import fmin
import numpy as np

```

```
6 def add_imp_vol_bs (tab_total):
7     def Calculate_ImpVol (t, d):
8         tentar = tab_total.loc[(tab_total['tau_d'] == d)]
9         tentar = tentar.loc[(tab_total['t'] == t)]
10        VecObsPrice = np.array(tentar['mid'])
11        VecSpot = np.array(tentar['under_mid'])
12        VecStrike = np.array(tentar['K'])
13        VecTau = np.array(tentar['tau'])
14        Vecr = np.array(tentar['r'])
15        Vecq = np.array(tentar['divyield'])
16
17        def BSM_price(S, K, T, sigma, r, q):
18            d1 = (np.log(S / K) + (r - q + (sigma ** 2) / 2) * T) / (
    sigma * np.sqrt(T))
19            d2 = d1 - sigma * np.sqrt(T)
20
21            call_price = S* np.exp(-q * T) *norm.cdf(d1) - K * np.exp(-r
     * T) * norm.cdf(d2)
22            return call_price
23
24        def fun(x):
25            soma = 0
26            for i in range(len(Vecr)):
27                soma += (VecObsPrice[i] - BSM_price(VecSpot[i],
    VecStrike[i], VecTau[i], x, Vecr[i], Vecq[i]))**2
28            return soma
29
30        spot = VecSpot.mean()
31        tau = VecTau.mean()
32        r = Vecr.mean()
33        strike = VecStrike.mean()
34        x0 = (2 * abs((np.log(spot / strike) + r * tau) / tau)) ** 0.5
35        return fmin(fun, x0, xtol=1e-8, ftol=1e-8, disp=False)
36
37    filtro = pd.DataFrame()
38    filtro ['t'],filtro['tau_d'] = tab_total['t'], tab_total['tau_d']
39    filtro.drop_duplicates(inplace = True)
40    indexx = []
41    for i in range(0,len(filtro['t'])):
42        indexx.append(i)
43    filtro ['index'] = indexx
44    filtro = filtro.set_index('index')
45
46    filtro['ImpVol_BS'] = 'NaN'
```

```
47
48      for i in range(len(filtro['t'])):
49          filtro['ImpVol_BS'][i] = Calculate_ImpVol (filtro['t'][i],
    filtro['tau_d'][i])[0]
```

A.4. Volatilidade Implícita, Skewness e Kurtosis - Corrado e Su

```
1  import pandas as pd
2  from scipy.stats import norm
3  import numpy as np
4  import scipy.optimize as optimize
5
6  def add_vol_skew_kurt (tab_total):
7      def Calculate_Skew_Kurt (t, d):
8          tentar = tab_total.loc[(tab_total['tau_d'] == d)]
9          tentar = tentar.loc[(tab_total['t'] == t)]
10         VecObsPrice = np.array(tentar['mid'])
11         VecSpot = np.array(tentar['under_mid'])
12         VecStrike = np.array(tentar['K'])
13         VecTau = np.array(tentar['tau'])
14         Vecr = np.array(tentar['r'])
15         Vecq = np.array(tentar['divyield'])
16
17         def BSM_price(S, K, T, sigma, r, q):
18             d1 = (np.log(S / K) + (r - q + (sigma ** 2) / 2) * T) / (
    sigma * np.sqrt(T))
19             d2 = d1 - sigma * np.sqrt(T)
20
21             call_price = S *np.exp(-q * T)* norm.cdf(d1) - K * np.exp(-r
     * T) * norm.cdf(d2)
22             return call_price
23         def corradosu (S, K, T, sigma, r, q, skewness, kurtosis):
24             d = (np.log(S / K) + (r - q + (sigma ** 2) / 2) * T) / (
    sigma * np.sqrt(T))
25             q3 = 1/6 * (S * sigma * np.sqrt(T))*((2*sigma*np.sqrt(T)-d)*
    norm.cdf(d) - sigma**2 * T* norm.cdf(d))
26             q4 = 1/24 * (S * sigma * np.sqrt(T))*((d**2 - 1 -3*sigma*np.
    sqrt(T)*(d-sigma*np.sqrt(T)))*norm.cdf(d)+sigma**3*T**(3/2)*norm.cdf(
    d))
27             corrado = BSM_price(S, K, T, sigma, r, q) + skewness*q3 + (
    kurtosis -3)*q4
28             #print(K, d, q3, q4,corrado)
29             return corrado
30         def fun(x):
31             v= x[0]
```

```
            skew = x[1]
            kurt = x[2]
            soma = 0
            for i in range(len(Vecr)):
                soma += (VecObsPrice[i] - corradosu(VecSpot[i],
    VecStrike[i], VecTau[i], v, Vecr[i], Vecq[i], skew, kurt))**2
            return soma

        spot = VecSpot.mean()
        tau = VecTau.mean()
        r = Vecr.mean()
        strike = VecStrike.mean()
        x0 = [(2 * abs((np.log(spot / strike) + r * tau) / tau)) ** 0.5,
     0.2, 0.2]
        result = optimize.minimize(fun, x0)
        return [result.x[0], result.x[1], result.x[2]]

    filtro = pd.DataFrame()
    filtro ['t'],filtro['tau_d'] = tab_total['t'], tab_total['tau_d']
    filtro.drop_duplicates(inplace = True)
    indexx = []
    for i in range(0,len(filtro['t'])):
        indexx.append(i)
    filtro ['index'] = indexx
    filtro = filtro.set_index('index')

    filtro['ImpVol_CS'] = 'NaN'
    filtro['Skewness'] = 'NaN'
    filtro['Kurtosis'] = 'NaN'

    for i in range(len(filtro['t'])):
        v, s, k = Calculate_Skew_Kurt (filtro['t'][i], filtro['tau_d'][i
    ])
        filtro['ImpVol_CS'][i] = v
        filtro['Skewness'][i] = s
        filtro['Kurtosis'][i] = k
```

A.5. Modelo de Black-Scholes-Merton

```
import numpy as np
from scipy.stats import norm

def rmse_blackscholes (S, K, r, q, sigma, tau, market):
    d1 = (np.log(S / K) + (r - q + 0.5 * sigma ** 2) * tau) / (sigma *
    np.sqrt(tau))
```

```
    d2 = d1 - sigma * np.sqrt(tau)
    call_price = S * np.exp(-q*tau) * norm.cdf(d1) - K * np.exp(-r * tau) * norm.cdf(d2)
    call_price = call_price.tolist()
    market_prices = market.tolist()
    rmse_bs = np.sqrt(np.mean((np.array(call_price) - np.array(market_prices)) ** 2))
    return rmse_bs
```

A.6. Modelo de Corrado e Su

```
import numpy as np
from scipy.stats import norm

def rmse_corradosu (S, K, r, sigma, tau, skew, kurt, market):
    d1 = (np.log(S / K) + (r + (sigma ** 2) / 2) * tau) / (sigma * np.sqrt(tau))
    d2 = d1 - sigma * np.sqrt(tau)

    call_price = S * norm.cdf(d1) - K * np.exp(-r * tau) * norm.cdf(d2)
    lambda_ = -0.5 * (skew * sigma * np.sqrt(tau) - (1/6) * (kurt - 3) * sigma**2 * tau)
    prices = call_price + lambda_ * (call_price / S) * (d1 - norm.cdf(-d1))

    prices = prices.tolist()
    market_prices = market.tolist()
    rmse_cs = np.sqrt(np.mean((np.array(prices) - np.array(market_prices)) ** 2))
    return rmse_cs
```

A.7. Método Support Vector Regression

```
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVR

def rmse_svr (X_train, y_train, X_test, y_test):
    svr = SVR(kernel='rbf', C=1, gamma=(len(X_train)**(-1)), tol=0.001, epsilon = 0.1)
    svr.fit(X_train, y_train)
    y_pred = svr.predict(X_test)
    rmse_svr = np.sqrt(mean_squared_error(y_test, y_pred))
    return rmse_svr
```

A.8. Método Random Forest

```
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor

def rmse_rf (X_train, y_train, X_test, y_test):
    rf = RandomForestRegressor(n_estimators=100, max_depth=20)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred))
    return rmse_rf
```

A.9. Método XGBoost

```
import numpy as np
from sklearn.metrics import mean_squared_error
import xgboost as xgb

def rmse_xgboost (X_train, y_train, X_test, y_test):
    xgb_model = xgb.XGBRegressor(learning_rate = 0.1, max_depth = 20)
    xgb_model.fit(X_train, y_train)
    y_pred = xgb_model.predict(X_test)
    rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred))
    return rmse_xgb
```

A.10. Método LightGBM

```
import numpy as np
from sklearn.metrics import mean_squared_error
import lightgbm as lgb

def rmse_lightgbm (X_train, y_train, X_test, y_test):
    lgb_model = lgb.LGBMRegressor(learning_rate = 0.1, max_depth = 20)
    lgb_model.fit(X_train, y_train)
    y_pred = lgb_model.predict(X_test)
    rmse_lgbm = np.sqrt(mean_squared_error(y_test, y_pred))
    return rmse_lgbm
```

A.11. Método Neural Networks

```
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
6
7 def rmse_nn (X_train, y_train, X_test, y_test):
8     scaler = MinMaxScaler()
9     X_train = scaler.fit_transform(X_train)
10    X_test = scaler.transform(X_test)
11    model = Sequential()
12    model.add(Dense(16, input_shape=(X_train.shape[1],), activation='
    relu'))
13    model.add(Dense(8, activation='relu'))
14    model.add(Dense(1, activation='linear'))
15    model.compile(loss='mean_squared_error', optimizer='adam')
16    model.fit(X_train, y_train, epochs=100, batch_size=32,
    validation_data=(X_test, y_test))
17    y_pred = model.predict(X_test)
18    rmse_nn = np.sqrt(mean_squared_error(y_test, y_pred))
19    return rmse_nn
```