**Machine Learning para Previsão de Mercado**

**Maria Eduarda P. Santana1, Julio L. R. Monteiro2, Arthur M. Rosa2, Paulo D’Angelo2**

1. Universidade de Yale – New Haven, Connecticut, Estados Unidos da América
2. B3 – Brasil Bolsa Balcão – São Paulo, São Paulo, Brasil

**Abstract:** As regras que regulam a dinâmica dos mercados e a sua capacidade preditiva são tópicos que geram debates intensos. As análises técnica e fundamentalista têm sido usadas ao longo das últimas décadas para encontrar padrões e tentar refutar a Hipótese do Mercado Eficiente[1]. As técnicas preditivas atuais usam algoritmos cada vez mais robustos, capazes de aprender em cenários caóticos e realizar previsões cada vez mais precisas. Este artigo analisa diferentes técnicas de *machine learning* (aprendizado de máquina) para previsão do comportamento do mercado e compara a sua eficiência com um algoritmo de passeio aleatório. A primeira técnica avaliada, ARIMA, é uma análise consagrada de séries temporais baseada em regressão linear e observação gráfica. A segunda técnica, LSTM, é algoritmo de *deep learning* (aprendizado profundo) baseado em redes neurais operando sobre dados de entrada contínuos. As comparações traçadas avaliam a performance desses algoritmos usando o *market data* (dados de mercado) público da B3 para instrumentos financeiros proeminentes: PETR4, ITUB4 e BVMF3. Os resultados indicam alto nível de imprevisibilidade do mercado durante o mês escolhido para avaliação (maio de 2017).

Palavras-chave: Mercado financeiro, market data, previsão de mercado, ARIMA, LSTM, B3, inteligência artificial, machine learning, deep learning.

1. **Introdução**

O debate intenso sobre a característica caótica ou aleatória dos preços das ações sofreu uma revolução na era da inteligência artificial. Em 1965, Eugene Fama apresentou a Hipótese do Mercado Eficiente, sugerindo que os preços correntes sempre refletiam todas as informações disponíveis[1]. De acordo com Fama, era impossível prever novas informações políticas, sociais e econômicas e, como resultado, o movimento do mercado era imprevisível. Contrário a essa teoria da aleatoriedade, surgiram teorias que caracterizam o mercado como um sistema complexo, caótico, que poderia, porém, ser previsto com base na análise determinística das interações entre fatores macro e microeconômicos sazonais[2]. Essa sinergia resultaria em um arranjo que poderia parecer aleatório, mas em realidade seguiria uma coletânea de regras. Atualmente, usando algoritmos avançados de machine learning e dados históricos, já é possível detectar padrões não-lineares remotos e questionar a imprevisibilidade do mercado. Por outro lado, um mercado que se comporta como um passeio aleatório é ainda uma hipótese aceita, especialmente para mercados emergentes e voláteis[3].

Um mercado caótico, ao contrário do aleatório, segue uma série de regras do tipo causa-e-efeito. Considerando a atmosfera, é muito difícil realizar previsões do tempo devido ao enorme número de variáveis envolvidas, dado que uma sutil alteração na estrutura inicial pode afetar o sistema como um todo de forma significativa – o chamado efeito borboleta[4]. Meteorologistas, porém, conseguem fazer previsões relativamente certeiras com base nas informações disponíveis. Isso não significa, porém, que pode começar a nevar no verão em países quentes. O estado anterior influencia o próximo, pois é um sistema determinístico que segue regras como a gravidade, relações químicas entre gases, física do vento e outros fatores que se associam. Esse é um sistema caótico. As regras são difíceis de entender, porém elas existem.

Por outro lado, jogar uma moeda é aleatório. Não existem regras escondidas interagindo se uma pessoa vai tirar cara ou coroa, existem chances iguais de obter qualquer um dos dois a cada vez que se joga uma moeda. É um fenômeno que pode ser entendido usando as regras probabilísticas. Ambos os sistemas caóticos e aleatórios parecem imprevisíveis, mas os caóticos só são assim devido ao modo como os fatores escondidos interagem. Esse é o ponto onde machine learning pode ajudar, encontrando as camadas escondidas nos relacionamentos entre os dados e tentando prever o que parecia desordenado.

Análises modernas sugerem que o mercado pode ser caótico ao invés de aleatório[5]; o que significa que um algoritmo inteligente poderia ser capaz de entender seus relacionamentos intrincados, identificar padrões e prever cenários futuros. O propósito deste artigo é estudar esta dicotomia comparando dois métodos de machine learning com um algoritmo de passeio aleatório e discutir os resultados no contexto do mercado de ações brasileiro.

A diferença entre machine learning e a programação tradicional é que em machine learning o computador aprende por meio de dados de treinamento e cria um programa capaz de prever a saída de dados. Por consequência, computadores conseguem aprender com os dados do passado e alterar a sua programação sem intervenção humana. Existem duas técnicas principais de machine learning: aprendizado supervisionado e aprendizado não supervisionado. No aprendizado supervisionado o computador recebe a entrada x e a saída esperada y, então tenta encontrar uma função f para f(x) = y, o que é chamado de aprendizado por indução. Por outro lado, no aprendizado não supervisionado, a máquina somente recebe a entrada x, que em geral não é categorizada previamente, e busca encontrar padrões nesses dados de entrada[6]. Para a previsão de preços de mercado ações o tempo (x) e o preço (y) existe uma correlação direta disponível, então este artigo escolheu explorar a técnica de aprendizado supervisionado.

Indo além, as classes de problemas resolvidas por machine learning se dividem em classificação, regressão, agrupamento, entre outros[7]. Em classificação, existe um número de categorias predefinidas para a saída. Por exemplo, seria possível fazer um algoritmo para predizer se vai chover e as respostas possíveis seriam “sim” e “não”. Por outro lado, num algoritmo de regressão um algoritmo para chuva tentaria prever a precipitação em centímetros, de forma que a saída seria um número ao invés de classes de saída. Ou seja, em problemas de classificação a função sendo aprendida é discreta, enquanto que na regressão a função é contínua[6].

O problema de previsão de preço no mercado de ações, da mesma forma que o exemplo da chuva, pode ser estudado de ambas as formas. Um algoritmo de classificação tentaria prever, por exemplo, se o preço das ações está com uma tendência de “subir” ou “descer” e um algoritmo de regressão tentaria prever um número que representa o preço daquela ação. As técnicas de machine learning estudadas especificamente neste artigo são o ARIMA – Auto-Regressive Integrated Moving Average (ou Médias Móveis Integradas e Auto Regressivas em português), que é um modelo preditivo bem estabelecido, e também o LSTM – Long Short Term Memory (ou Memória de Longa e Curta Duração em português), um algoritmo de redes neurais recorrentes. O modelo ARIMA é usado para previsões baseadas em regressão linear e o algoritmo LSTM também foi configurado para gerar uma saída numérica (regressão) para fins de comparação.

1. **Dados e pré-processamento**

Os dados usados neste artigo são parte das séries históricas de Market Data providos gratuitamente pela B3, no seu site, por meio de um servidor FTP. As séries temporais são os dados de uma sessão de negociação do mercado de ações, em arquivos separados por dia .

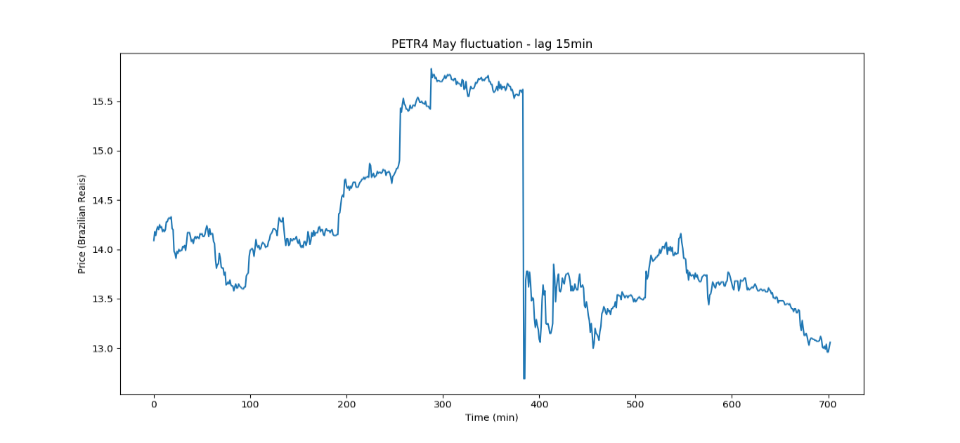


Figura 1 – Flutuação de PETR4 em maio de 2017

Os dados foram pré-processados, extraindo os dados de um pequeno conjunto de alguns instrumentos selecionados. Os instrumentos escolhidos são altamente negociados no mercado brasileiro de ações e a liquidez de cada um deles mostra quão rapidamente os algoritmos podem detectar mudanças de momento econômico. PETR4 é a ação preferencial da Petróleo Brasileiro AS (Petrobras), a maior empresa de petróleo e gás natural do Brasil; ITUB4 é a ação preferencial do banco Itaú Unibanco, o maior banco privado na América Latina; e BVMF3 é a ação ordinária da B3, a quinta maior bolsa de valores do mundo em termos de valor[9].

Os preços negociados foram fixados em intervalos de 15 minutos (houveram testes também com intervalos de 10 min e 5 min) para o mês de maio de 2017. Assim, houveram por volta de 700 pontos de preço para cada instrumento, alimentados para os algoritmos, por meio de um arquivo. Para desenvolver as simulações foi usado o ambiente Anaconda[10], usando a linguagem Python e as bibliotecas TensorFlow e Keras para implementação da rede neural.

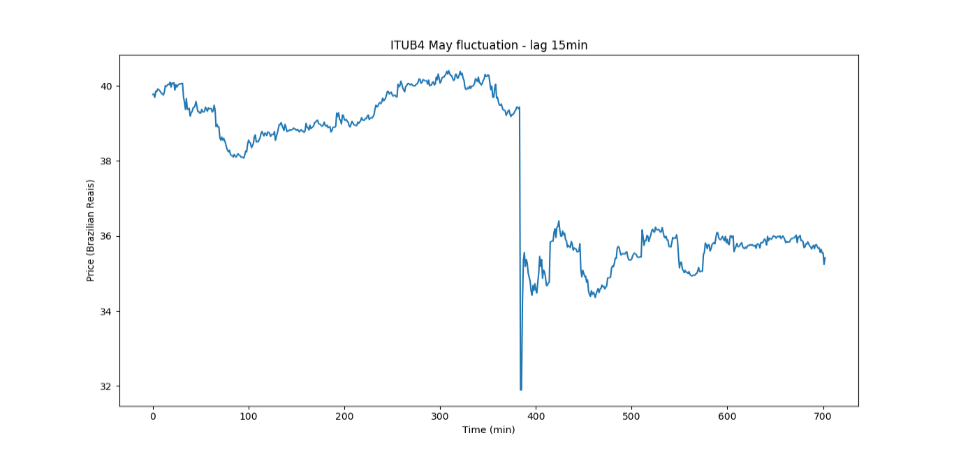


Figura 2 – Flutuação de ITUB4 em maio de 2017

Os instrumentos PETR4, ITUB4 e BVMF3 mostraram padrões temporais de preço similares, indicando uma correlação do preço dessas ações no mercado brasileiro. Além do mais, a queda significativa de preço em todos os gráficos mostra como eventos políticos e econômicos externos tiveram importante papel na governança corporativa em maio de 2017.

A queda generalizada de preços observada nos gráficos aconteceu por volta de 18 de maio, quando o índice IBOVESPA (IBOV), principal indicador do mercado teve uma baixa de 10,47% comparando com fechamento do dia anterior[11]. Na noite anterior, o presidente da empresa JBS, Joesley Batista, havia divulgado na grande imprensa um áudio associando o então presidente do Brasil, Michel Temer, a um esquema de propina e corrupção. O índice IBOV engloba os 50 papéis mais negociados na B3, que juntos representam 80% do valor negociado no Brasil. Por consequência, a queda significativa do índice devida à turbulência política resultou em uma interrupção de 30 minutos no mercado, para conter a alta volatilidade que se apresentou. Como resultado, este artigo também busca avaliar como algoritmos diferentes lidam com mudanças bruscas no mercado.

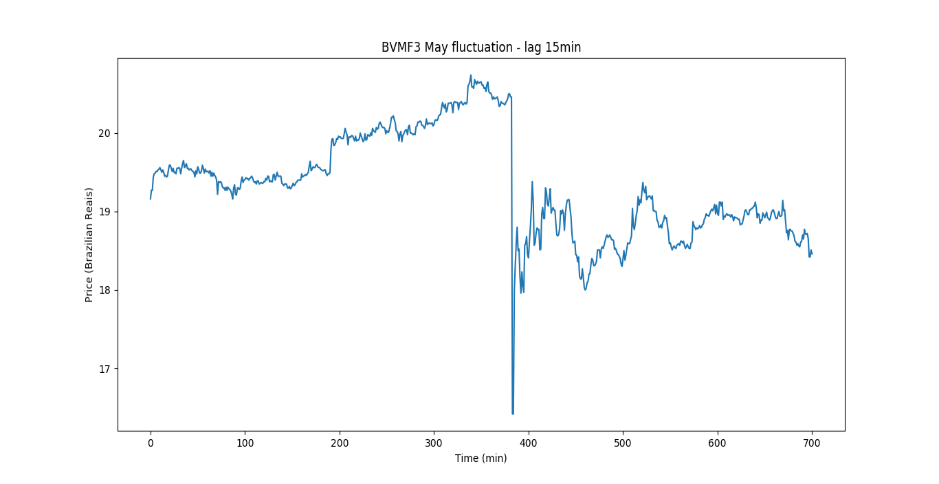


Figura 3 – Flutuação de BVMF3 em maio de 2017

1. **Estacionarização**

Para ser capaz de realizar previsões é necessário encontrar a componente invariante no tempo de uma série e projetar para tempos futuros. Se o comportamento do mercado fosse completamente diferente a cada dia seria impossível predizer o próximo cenário. Porém, se houver um comportamento constante ao longo do tempo, surge um padrão que pode ser explorado. Uma série estacionarizada, seguindo esse princípio, é uma série com média constante, variância constante e covariância constante ao longo do tempo. O ARIMA exige uma série estacionarizada como entrada, então, a título de se observar a correspondência, a mesma série foi usada para o LSTM. Se a série temporal inicial não for estacionária – que costuma ser o caso para problemas reais – é necessário transformá-la usando logaritmo, diferenciação, deflação, decomposição ou outras transformações estabilizantes[12].

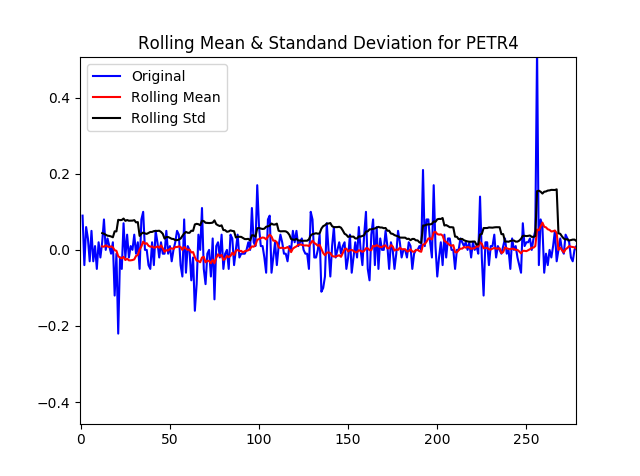
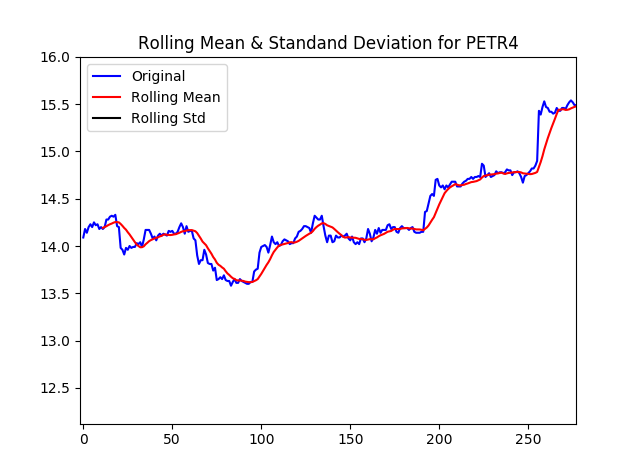


Figura 4 – Comparação entre uma série não estacionária (esquerda) e estacionária (direita)

As transformadas logarítmicas são geralmente usadas para séries com tendências crescentes. Entretanto, como as séries temporais dos instrumentos usados não tem uma característica clara de tendência ao longo do mês de maio, foi aplicada a diferenciação sem usar a transformação logarítmica antes. O melhor jeito para se identificar se uma série é estacionária depois da transformação é usar o teste de Dickey-Fuller Aumentado[13], que está disponível na biblioteca *statsmodel* do Python. Esse teste verifica a hipótese nula, que, quando aceita, significa que a série temporal tem uma raiz unitária e não é estacionária, e se rejeitada, indica que a série não tem uma estrutura dependente de tempo e é estacionária[14].

As series parciais de PETR4 antes e depois da estacionarização estão na Figura 4, permitindo-se observar os efeitos da transformação estabilizadora.

1. **ARIMA (Autoregressive Integrated Moving Average)**
2. Algoritmo

ARIMA[8], também conhecido como método Box-Jenkins, é um dos modelos mais tradicionais de machine learning para predição de series temporais. Uma série temporal é uma sequência de pontos indexados em um certo intervalo temporal. Nessa etapa do estudo, os preços das ações foram indexados com um intervalo de 15 minutos.

O modelo combina um componente auto regressiva (AR), que observa a relação entre os números anteriores; uma componente integrativa (I), que usa diferenciação para tornar a série temporal estacionária e uma componente de médias móveis (MA), que observa a dependência entre os erros residuais anteriores. Esses elementos são enviados respectivamente como parâmetros para a função ARIMA(p,q,d).

*Parâmetros (p,d,q)*

Existem basicamente duas maneiras de definir os parâmetros. A primeira é o método interativo, no qual o programa testa uma série de números e escolhe quais obtém um resultado mais otimizado. Porém essa abordagem, quando usada isoladamente é lenta e exige grande quantidade de poder computacional. A segunda abordagem é a definição dos parâmetros por meio da análise dos gráficos de correlação (ACF) e correlação parcial (PACF). Existem certas regras para interpretar os gráficos e, na maioria das vezes, algumas variações podem ser testadas para encontrar o melhor conjunto de parâmetros.

A forma mais fácil de determinar ‘p’ (componente auto regressiva) e ‘q’ (componente das médias móveis) é usar ‘p’ como o número de intervalos no qual o gráfico PACF chega a zero pela primeira vez, onde ‘q’ é o número de intervalos no qual o gráfico ACF cruza a linha de significatividade estatística pela primeira vez[15]. Uma abordagem mais detalhada pode ser encontrada nas notas de análise estatística preditiva do professor Robert Nau[12]. No corrente artigo optou-se por uma abordagem híbrida. A análise gráfica foi realizada e então seguida de uma análise interativa de certas combinações de parâmetros para garantir uma resposta mais otimizada.

A ordem de diferenciação ‘d’ raramente é maior do que 2 em algoritmos financeiros e a superdiferenciação pode introduzir níveis de dependências desnecessários[16]. É uma situação na qual o algoritmo olha muito para o passado e encontra padrões que não são compatíveis com a dinâmica acelerada do mercado. Sendo assim, este estudo testou apenas a ordem 1 de diferenciação (1 intervalo). Após configurar os parâmetros e testar o algoritmo é uma boa prática avaliar o gráfico de ACF para encontrar erros residuais para certificar que todas as tendências foram identificadas e os residuais são apenas ruído branco (valores não significativos).

1. Implementação

Para implementar o algoritmo é necessário dividir o conjunto de dados em dados para treinamento e teste. O modelo usa os dados de treinamento para aprender um padrão progressivamente e os dados de teste para avaliar qual foi o grau de aprendizado. A massa de dados foi dividida na proporção de 65% para treinamento e 35% para teste.

Os parâmetros ótimos (p,d,q) foram ligeiramente diferentes para cada ação, mas o processo usado para encontrar esses parâmetros foi equivalente. Para demonstrar a análise gráfica, foi usado o instrumento PETR4.

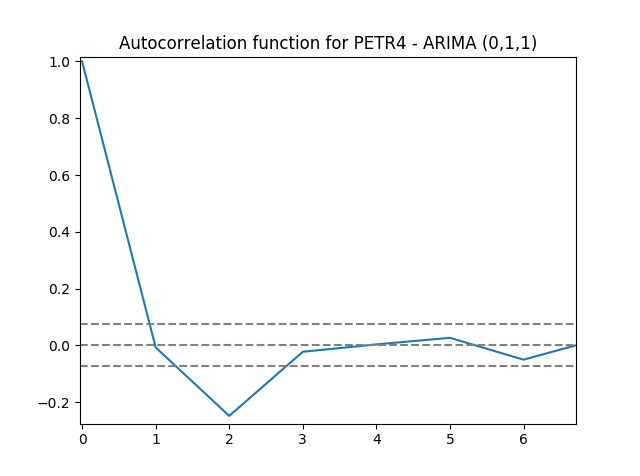
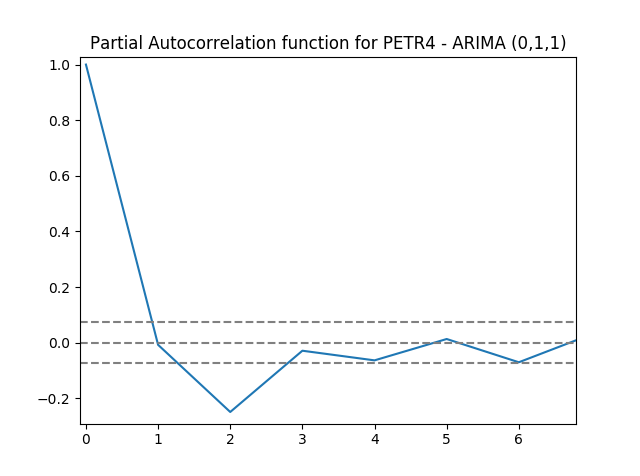
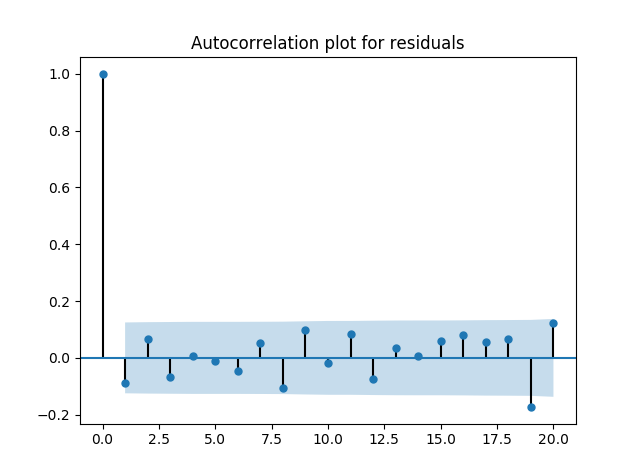


Figura 5 – gráficos ACF e PACF para PETR4

Figura 6 – Gráfico ACF para o erro residual de PETR4

Examinando os gráficos de auto correlação e auto correlação parcial para PETR4 (Figura 5), observa-se que a linha da auto correlação cruza a linha de significância estatística por volta do valor 1 e a auto correlação parcial também chega a zero por volta do valor 1, sendo assim, foram executadas as interações para ARIMA(0,1,1), ARIMA (1,1,0) e ARIMA(1,1,1); a primeira apresentou um resultado sutilmente melhor, como se pode observar na Figura 6, avaliando o gráfico de auto correlação dos residuais para ARIMA(0,1,1). Como foi dito, os erros residuais não têm correlação estatística significativa (todos estão dentro da área sombreada), portanto o algoritmo está parametrizado de forma otimizada.

1. Raiz Quadrada do Erro Quadrático Médio (RMSE - Root Mean Squared Error)

A raiz quadrada do erro quadrático médio (RSME – Root Mean Square Error) foi usada para medir a precisão do algoritmo, representando o desvio padrão dos erros previstos (diferença entre os valores previstos e os valores de teste). Um procedimento similar ao descrito nas seções A e B deste capítulo foi utilizado para os outros instrumentos (ITUB4 e BVMF3) e os RMSEs resultantes foram compilados na Tabela 1 abaixo:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **ARIMA (0,1,1)** | **ARIMA(1,1,0)** | **ARIMA(1,1,1)** |
| PETR4 | 0.04934 | 0.04935 | 0.04958 |
| ITUB4 | 0.10901 | 0.10882 | 0.11225 |
| BVMF3 | 0.07148 | 0.07112 | 0.07419 |

Tabela 1 – Comparação do RSME entre todos os modelos de ARIMA testados para cada instrumento

Como observado, os RMSEs são bastante similares para p = 0 🡪 1 e q = 0 🡪 1, que indica que a forma com que os valores temporais dos instrumentos observados (parte auto regressiva do algoritmo) é semelhante à forma como seus erros residuais se correlacionam (parte do algoritmo referente às médias móveis).

1. Gráficos das previsões

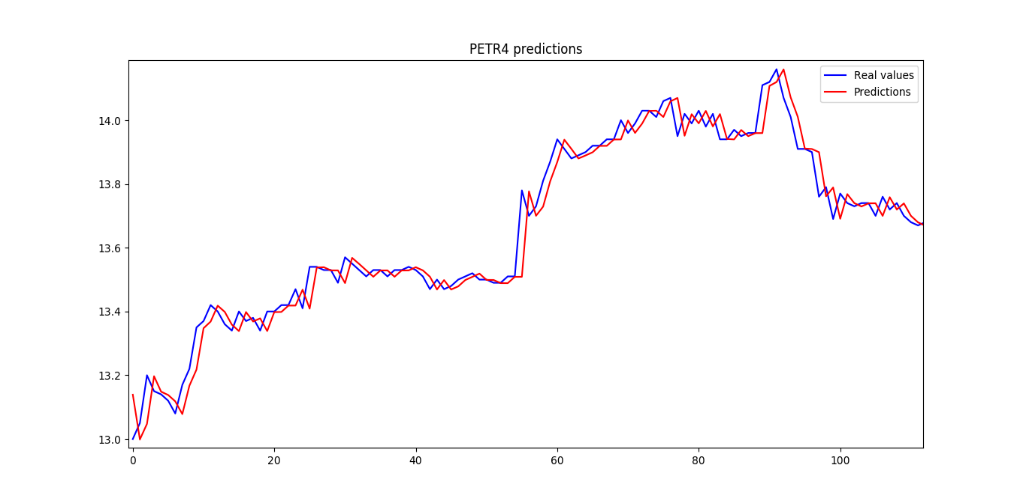
Os gráficos das previsões foram produzidos usando os melhores parâmetros para cada ação (de acordo com a Tabela 1). O gráfico para PETR4 (Figura 7) é um exemplo de como foi a previsão para os primeiros 100 preços usando o modelo ARIMA (0,1,1).

Figura 7 – Previsões para PETR4 no mês de maio de 2017

1. **LSTM (Long Short Term Memory – Memória de Longa e Curta Duração)**
2. Algoritmo

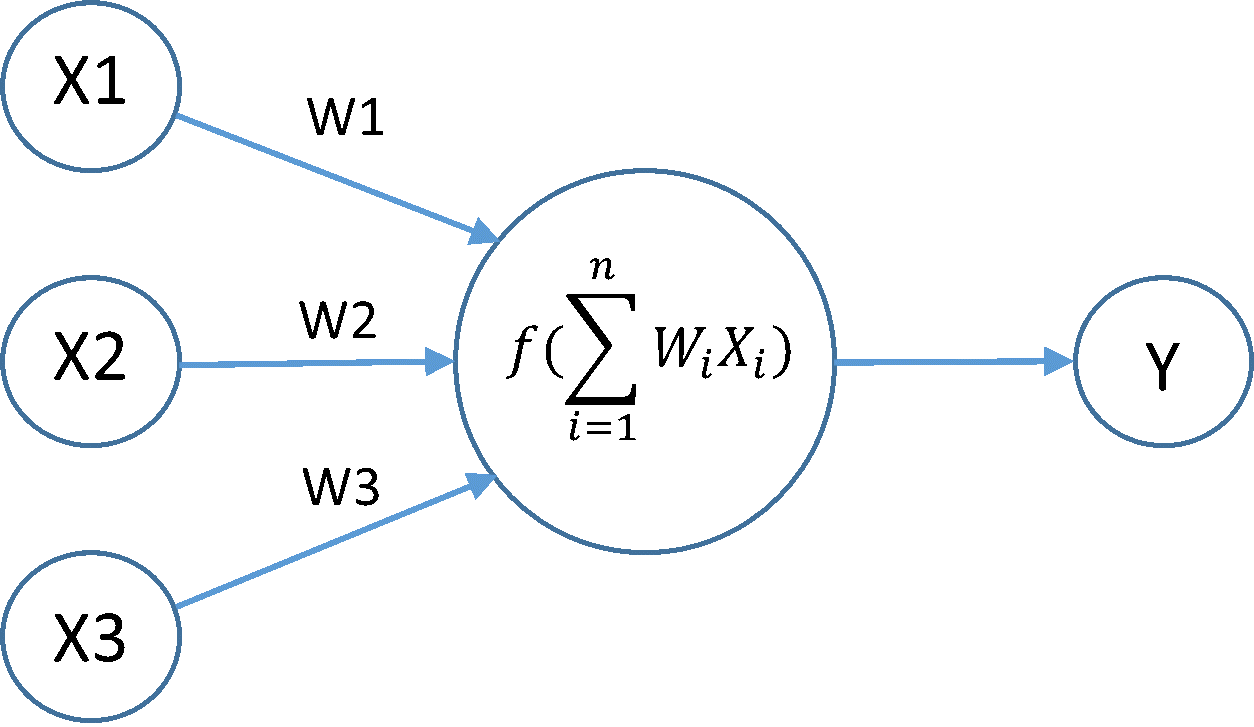
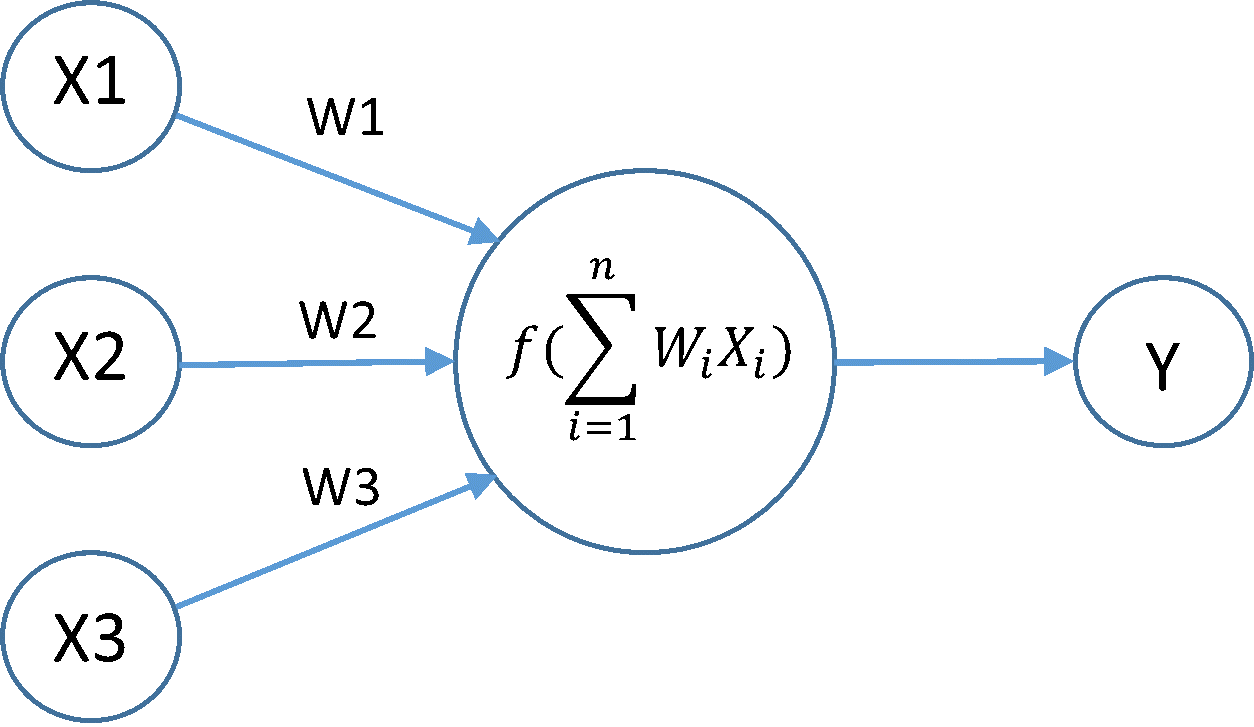
Redes Neurais Artificiais são modelos matemáticos inspirados no funcionamento das redes neurais biológicas. Elas são compostas de unidades interconectadas chamadas de neurônios artificiais, que processam dados recebidos nas suas entradas usando uma função de ativação e produzindo um dado de saída. Na sua forma mais tradicional, os neurônios são dispostos em camadas, sendo que a camada que se conecta com os dados de entrada é chamada de camada de entrada, seguida por diversas camadas intermediárias de neurônios, chamadas camadas escondidas, que por fim se conectam com uma camada de saída, de neurônios que trazem os dados processados para fora da rede. As conexões entre os neurônios têm um peso, que indica o quanto a ativação do neurônio anterior deve influenciar a ativação do neurônio posterior.

As arquiteturas recentes de redes neurais que contam com diversas camadas escondidas deram origem ao termo *“deep learning”*, ou aprendizado profundo. Recentemente o deep learning ganhou bastante popularidade, sendo um dos algoritmos mais populares na inteligência artificial. Esse tipo de algoritmo é usado intensamente em aplicações de visão computacional, reconhecimento de fala, processamento de imagens e previsão.

Como fortes ferramentas para identificação e padrões, as redes neurais foram decisivas para os que acreditam que o mercado não é aleatório, mas sim caótico. Portanto, a capacidade de uma rede neural de capturar padrões de comportamento remotos a torna uma ferramenta adequada para modelar sistemas caóticos que aparentam funcionar por meio de regras ocultas[5]. Porém, a previsão no mercado de ações é bastante baseada em dados observados historicamente, assim as redes neurais tradicionais onde os dados fluem sempre da entrada para a saída (feed forward) não são ideais, por não conseguirem guardar as informações temporais necessárias para enxergar comportamentos sazonais.

Porém, existe um tipo especializado de rede neural dotado de uma memória de estado, chamado rede neural recorrente (RNN – Recurrent Neural Network), que é frequentemente utilizado na análise de padrões em séries temporais. A diferença principal entre a rede tradicional e a RNN é a estrutura do neurônio artificial (Figura 8).

Figura 8 – Neurônio tradicional (esquerda) e o neurônio RNN (direita)



O neurônio é uma unidade onde as entradas (x1, x2 etc.) são processadas aplicando pesos (w1,w2 etc.) para determinar a relevância de cada entrada para a geração da saída[17]. No lado esquerdo da Figura 8, a soma dos produtos das entradas e dos pesos é usada como parâmetro para uma função de ativação *f* dentro do neurônio – que pode ser uma função linear, sigmoide, tangente hiperbólica ou outras funções que sejam apropriadas para o conjunto de dados. Por outro lado, em uma rede RNN, o neurônio não só recebe as entradas x1, x2, etc., mas também recebe o resultado de ativação da interação passada (multiplicada por um peso, que indica qual a relevância da informação do passado para o presente), permitindo que a informação do “estado” do passado seja mantida.

Esse tipo de redes neurais recorrente é usado especificamente em problemas com entradas contínuas, uma vez que é capaz de processar a informação de forma progressiva sem dispersar os pesos, de camada em camada. Porém esse elo com passado só permite fazer referência a um intervalo anterior. Para ter acesso a um passado mais remoto é preciso criar laços cada vez mais complexos, desdobrando a rede e multiplicando a quantidade de neurônios e camadas, arriscando perder o valor dos pesos, quando se multiplica sucessivamente os seus valores (estouro de gradiente ou gradiente zerado)[19].

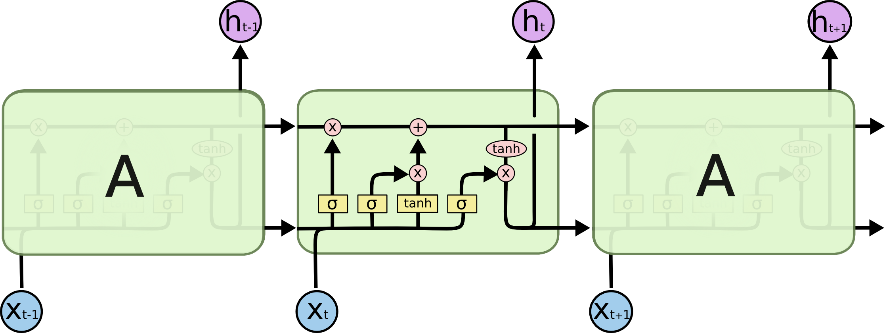
Um tipo específico de rede neural recorrente que evita esse problema de estouro de gradiente é chamado LSTM (Long Short Term Memory, ou Memória de Longa e Curta Duração) que utiliza um modelo de neurônio mais complexo com portas aditivas (ao invés de multiplicativas) para seletivamente lembrar ou esquecer de informações do passado.

Figura 9 – Neurônio LSTM

Uma descrição mais detalhada do neurônio LSTM (Figura 9) revela quatro portas internas – três delas usando funções sigmoides e uma usando tangente hiperbólica – responsáveis por atualizar o estado da célula e passar adiante [18]. O diagrama é semelhante ao da rede recorrente (Figura 8, a direita), com a diferença que as entradas são posicionadas na parte inferior e as saídas ficam na parte superior, para aumentar a clareza. A diferença fundamental entre a rede RNN clássica e a LSTM reside no fato que essa última usa as citadas portas internas de forma aditiva, somando os valores dos pesos do passado ao invés de multiplicá-los, minimizando o problema do estouro de gradiente.

O algoritmo escolhido para este trabalho usa o modelo LSTM com duas camadas escondidas.

1. Implementação

Da mesma forma que no modelo ARIMA, é necessário dividir o conjunto de dados em dados de treinamento e teste. Para manter uniformidade e facilitar a comparação, foi usada a mesma proporção de 60% para treinamento e 35% para teste. A distribuição inicial de pesos na rede é aleatória e eles vão gradualmente sendo refinados conforme o algoritmo é treinado. Além disso, o método de treinamento da rede é o BPTT (Backpropagation through time, ou Retro propagação ao longo do tempo), uma técnica baseada em descida de gradiente que minimiza a função custo, propagando os erros residuais e ajustando os pesos de trás para frente (da saída para a entrada)[20].

O treinamento é dividido em *epochs*, ou épocas, que representam a exposição de todos os dados de treinamento ao algoritmo. Poém, o modelo não precisa consumir todos os dados de uma só vez, podendo fazer o aprendizado em levas menores (batch learning), atualizando os pesos da rede a cada leva, não somente no final da epoch. É possível usar os dois métodos para dividir a massa de treinamento: mini-batches e descida estocástica de gradiente.

Quando se usa mini-batches, o tamanho da leva é pré-definido e os dados de treinamento são divididos seguindo o mesmo padrão. Por exemplo, se houverem 500 pontos para treinamento, se o tamanho da leva foi 250, cada epoch irá contém 2 ciclos de 250. Ao invés de atualizar os pesos depois que cada epoch termina, o algoritmo atualiza os pesos depois de cada iteração, permitindo atingir um ponto ótimo da solução mais rapidamente. Enquanto isso, usando a descida de gradiente estocástica, na realidade é o mesmo que usar o tamanho de leva de 1, de forma que o algoritmo atualiza os pesos da rede a cada novo exemplo de treinamento.

Decidir a melhor estratégia para montar as levas é uma parte importante nos algoritmos LSTM. Usar um tamanho de leva muito grande exige mais memória, porém usar números pequenos pode levar a baixa precisão se o algoritmo não estiver programado para guardar o estado entre levas[21]. O método estocástico de descida de gradiente foi usado neste trabalho pois, como no ARIMA, o objetivo era fazer previsões de 1 intervalo apenas – prever o próximo preço das ações depois de cada ponto. Outro ponto que sustenta essa escolha é a intenção de conservar memória depois de cada iteração, sendo que o parâmetro “**stateful**” do framework Keras, usado na elaboração da rede LSTM foi configurado como ativo, indicando que oe stado dos neurônios deve ser preservado depois de cada leva de treinamento, ao invés de ser reiniciado.

Em adição ao tamanho da leva, o número de epochs é, da mesma forma, um parâmetro importante na definição do modelo de redes neurais LSTM. Depois de caad epoch, o estado da célula é reiniciado, e todos os dados são apresentados novamente, preservando apenas os pesos entre os neurônios. Se o número de epochs for pequeno demais o algotirmo pode não ter oportunidade de encontrar os pesos otimizados para conseguir fazer previsões futuras adequadas durante a fase de teste. Entretanto, se o número for grande demais o modelo pode apresentar *overfitting*, ou seja, aprender os exemplos de treinamento “de cór”, de modo que, quando for exposto aos exemplos de teste, gere resultados muito distantes dos esperados – o algoritmo aprende todo o ruído dos dados de treinamento e não dá bons resultados em outros dados[22]. O número de epochs escolhido para este trabalho foi de 50, que, depois de testes experimentais mostrou-se razoável para os conjuntos de dados de 700 pontos dos preços das ações escolhidas.

Outra característica que também poderia ser responsável por overfitting seria o número de neurônios nas camadas escondidas. Existem regras gerais sobre isso exploradas no livro de Jeff Heaton[23], que sugerem usar um número de neurônios próximo ao número de entradas e saídas. Ainda assim, como comenta o autor, as regras são apenas um ponto de partida, sendo importante realizar testes iterativos. Para o caso estudado, cada entrada e saída tinham apenas um elemento em cada iteração, foram testados alguns valores pequenos e o número mais adequado foi de 3 neurônios por camada.

Uma vez que as condições iniciais de pesos eram geradas aleatoriamente, o algoritmo gera resultados ligeiramente diferentes a cada execução. Para diminuir essa componente aleatória, o programa foi executado 10 vezes e foi levada em conta a média dos RMSEs para comparar com os resultados do ARIMA.

1. Raiz Quadrada do Erro Quadrático Médio (RMSE - Root Mean Squared Error)

Da mesma forma que no ARIMA, a raiz quadrada do erro quadrático também foi utilizada para avaliar a precisão do algoritmo. Alguns parâmetros foram alterados manualmente para verificar a existência de overfitting e o efeito do número de camadas escondidas.

|  |  |  |
| --- | --- | --- |
|  | **RMSE de Testes** | **RMSE de Treinamento** |
| PETR4 | 0.04933 | 0.00976 |
| ITUB4 | 0.11141 | 0.02689 |
| BVMF3 | 0.07356 | 0.01891 |

Tabela 2 - Comparação entre os erros obtidos no treinamento e teste

A diferença entre os erros de treinamento e teste podem ser explicados pela origem dos dados. O conjunto de dados de treinamento foi examinado repetidas vezes pelo algoritmo durante o treinamento; já os dados de teste eram avaliados apenas uma vez, durante a última etapa de cálculo de precisão, medindo a capacidade de generalização do algoritmo. Como resultado, espera-se que o erro nessa etapa de testes seja maior. Entretanto, o RMSE de testes oferece uma estimativa melhor para entender como o algoritmo iria se comportar num cenário de tempo real. Se a diferença fosse ainda maior, poderia ser um sinal de overfitting – quando a rede se comporta muito melhor durante o treinamento e vai mal na parte do teste.

O número de camadas escondidas foi reduzido para um para testar a relevância de usar uma segunda camada durante o treinamento, como nesse caso, como usa-se um modelo simples de LSTM e não envolve deep learning. Foi gerada uma tabela (Tabela 3) comparando os resultados com uma ou duas camadas.

|  |  |  |
| --- | --- | --- |
|  | **2 camadas** | **1 camada** |
| PETR4 | 0.04933 | 0.04949 |
| ITUB4 | 0.11141 | 0.11056 |
| BVMF3 | 0.07356 | 0.07440 |

Tabela 3 – Comparação entre os erros com números diferentes de camadas escondidas

Mesmo com o modelo de 2 camadas tendo mostrado resultados sutilmente melhores os erros em geral são muito parecidos, sugerindo que a adição de uma segunda camada pode não afetar significativamente os resultados para a massa de dados escolhida. Talvez se a massa de dados fosse maior, poderia ser possível observar impactos maiores na adição de uma segunda camada para tentar encontrar padrões escondidos.

1. Gráficos das previsões

Foram plotados os gráficos das previsões para cada ação, sendo que o gráfico de PETR4 (Figura 10) foi usado como exemplo da previsão dos 100 primeiros preços com o modelo de 1 camada.

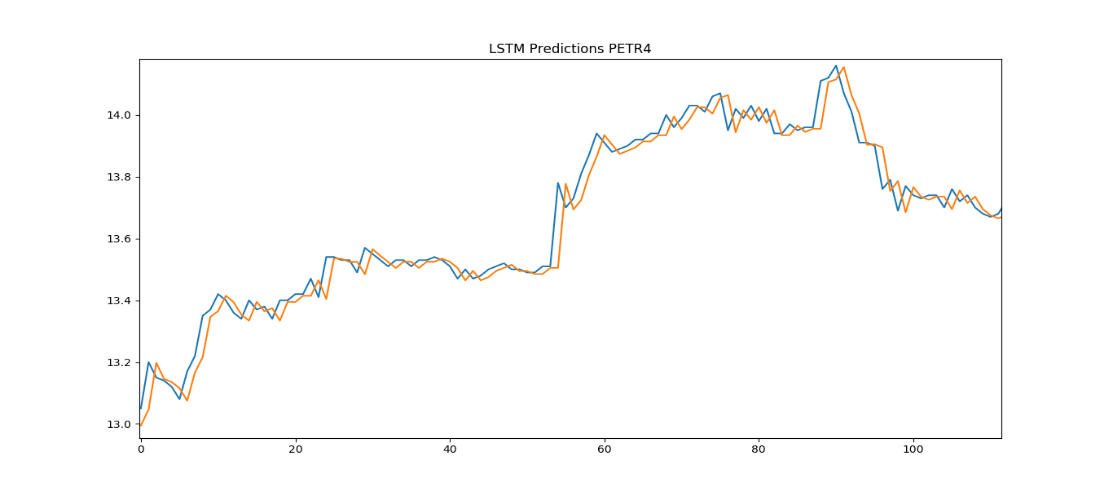


Figura 10 – Previsões de PETR4 usando LSTM para maio de 2017

1. **Análise comparativa dos algoritmos**

Como observado nas tabelas 1, 2 e 3, o RMSE fica realmente similar para ambos os algoritmos, sendo que o LSTM ou ARIMA tem uma performance ligeiramente melhor dependendo do instrumento. Uma análise do erro simples em centavos usando o gráfico de Whisker[24] foi realizada para comparar a performance dos algoritmos (Figuras 11, 12 e 13). Além disso, comparamos a performance desses algoritmos com uma linha de base, que seria o algoritmo de travessia aleatória (Random Walk) para testar a característica de imprevisibilidade do mercado para a massa de dados escolhida.

|  |  |  |  |
| --- | --- | --- | --- |
| **PETR4** | ARIMA | LSTM | Random Walk |
| Média | 0.032137 | 0.033582 | 0.032152 |
| Desvio Padrão | 0.037512 | 0.036570 | 0.037526 |
| Min | 0.000173 | 0.000836 | 0.000114 |
| Max | 0.271152 | 0.279962 | 0.271137 |

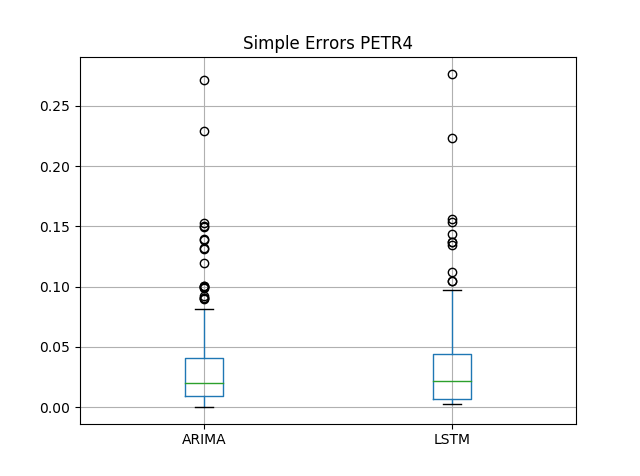


Tabela 4 – Erros de PETR4 para cada modelo

Figura 11- Erros Simples de PETR4

|  |  |  |  |
| --- | --- | --- | --- |
| **ITUB4** | ARIMA | LSTM | Random Walk |
| Média | 0.070288 | 0.073057 | 0.070199 |
| Desvio Padrão | 0.082618 | 0.082522 | 0.082526 |
| Min | 0.000104 | 0.001414 | 0.000973 |
| Max | 0.728268 | 0.737137 | 0.728490 |

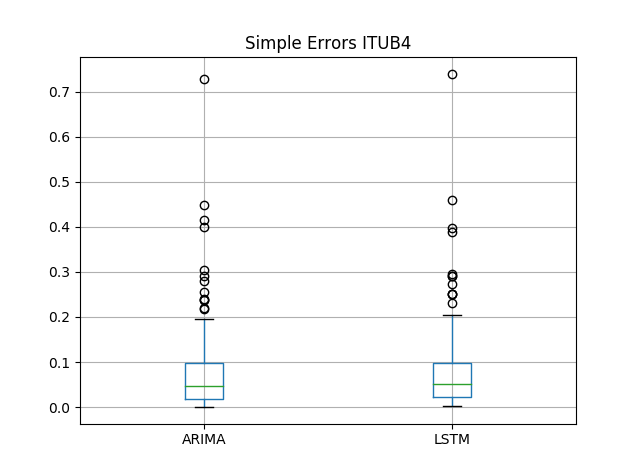
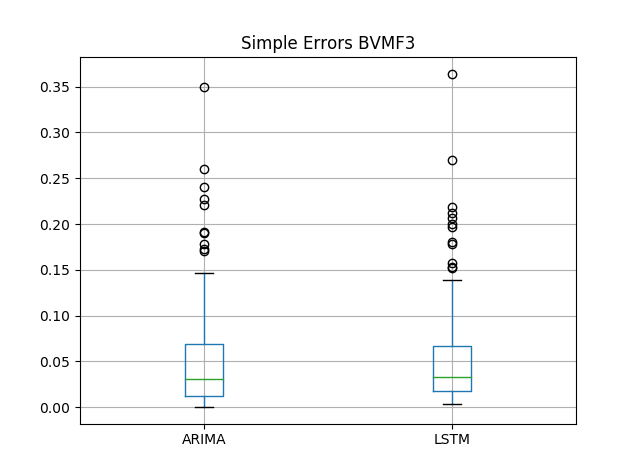


Tabela 5 – Erros de ITUB4 para cada modelo

Figura 12- Erros Simples para ITUB4



|  |  |  |  |
| --- | --- | --- | --- |
| **BVMF3** | ARIMA | LSTM | Random Walk |
| Média | 0.049130 | 0.052281 | 0.048930 |
| Desvio Padrão | 0.051637 | 0.051110 | 0.051528 |
| Min | 0.000253 | 0.002413 | 0.000066 |
| Max | 0.349344 | 0.368737 | 0.350923 |

Tabela 6 – Erros de BVMF3 para cada modelo

Figura 13- Erros Simples para BVMF3

O algoritmo de travessia aleatória foi construído simplesmente usando ARIMA(0,1,0), um modelo no qual não existe auto correlação entre os termos e nem entre os seus erros residuais. A próxima observação será sempre igual à anterior, usando um incremento ou decremento aleatório[25]. Esse método dá substância à Hipótese do Mercado Eficiente, já que diz que o movimento do mercado é aleatório e não pode ser previsto. Como observado, em dois dos três instrumentos analisados tiveram resultados melhores usando o algoritmo de travessia aleatória do que usando os modelos de machine learning.

1. **Conclusão**

Os resultados deste trabalho sugerem que machine learning pode ser aplicado no mercado financeiro e apresentar bons resultados, mesmo o cenário do período utilizado no estudo tendo apresentado alta volatilidade e imprevisibilidade. É importante ser ressaltado que os papéis analisados no trabalho estão inseridos num período de turbulência política. Consequentemente, os resultados podem, de fato, sugerir que a travessia aleatória traz melhores resoltados, indicando que a predição é muito difícil utilizando-se apenas dados referentes ao período de um mês.

Os resultados também podem ser utilizados para comparar os modelos ARIMA e LSTM. Os erros (RMSEs) foram bastante similares para ambos os algoritmos. Embora o modelo ARIMA seja mais rápido e demande menos poder e tempo de processamento, este requer uma parametrização cuidadosa e uma quantidade significativa de análises de gráficos. Por outro lado, o modelo LSTM se adapta por si só durante o processo de ajuste de pesos na etapa de treinamento do modelo, sendo necessário apenas definir o número máximo de *epochs* de treinamento, o tamanho dos lotes e outros parâmetros gerais de configuração inicial da rede neural que podem ser otimizados, onde neste trabalho adotou-se a estratégia de exploração das combinações e análise de resultado para escolha da configuração com melhor resultado final.

Dentre os trabalhos futuros possíveis é possível elencar alterações no modelo, como por exemplo, aumentar a base de dados de preços para incluir mais meses ou até anos, para permitir que os modelos encontrem padrões mais complexos. Além disso, ao invés de apenas levar em conta os preços históricos dos instrumentos selecionados para o trabalho, seria possível correlaciona-los com outros parâmetros que poderiam ser alimentados aos modelos, como momento, volatilidade, preços de índices ou o valor do câmbio. Ou ainda, os parâmetros do modelo LSTM poderiam ser reajustados (aumento do número de epochs) ou uma simulação com janela móvel poderia ser utilizada: em outras palavras, o modelo poderia ser treinado com os dados do mês de janeiro, testado com fevereiro e utilizado para prever os dados de março. Na sequência, a janela de dados se moveria, os dados de fevereiro seriam utilizados para teste e o modelo preveria os preços para abril, e assim por sucessivamente.

Foi observado como aprendizado de máquina pode ser utilizado para predição de preços de ações e como o mercado financeiro brasileiro se comporta com relação ao dilema de randômico versus caótico. Como a maioria dos países emergentes, o comportamento é inconstante e mais dados poderiam ser levados em conta para estas flutuações[3]. Muitas instituições financeiras estão se aprimorando no uso de técnica baseadas em machine learning para lidar com a incrível quantidade de dados produzida atualmente. O desafio, agora, é encontrar os melhores algoritmos e organizar os dados para se adequar a cenários políticas e econômicos específicos.

Os dados e o códigos-fonte utilizados e produzidos neste trabalho estão disponíveis em [26].

1. **Referências**

[1] LO, A. W.. Efficient markets hypothesis. 2007.

[2] HSIEH, D. A. Chaos and nonlinear dynamics: application to financial markets. **The journal of finance**, v. 46, n. 5, p. 1839-1877, 1991.

[3] ZEREN, F.; KONUK, F. Testing the random walk hypothesis for emerging markets: evidence from linear and non-linear unit root tests. **Romanian Economic and Business Review**, v. 8, n. 4, p. 61, 2013.

[4] LORENZ, E. The butterfly effect. **WORLD SCIENTIFIC SERIES ON NONLINEAR SCIENCE SERIES A**, v. 39, p. 91-94, 2000.

[5] LAWRENCE, R. Using neural networks to forecast stock market prices. **University of Manitoba**, 1997.

[6] BROWNLEE, J. Basic Concepts in Machine Learning - Machine Learning Mastery. Machine Learning Mastery. Disponível em: <http://machinelearningmastery.com/basic-concepts-in-machine-learning/>. Acessado: 01/08/2017.

[7] MADGE, S. Predicting Stock Price Direction using Support Vector Machines. **Independent Work Report Spring**, 2015.

[8] PAI, P-F; LIN, C-S. A hybrid ARIMA and support vector machines model in stock price forecasting. **Omega**, v. 33, n. 6, p. 497-505, 2005.

[9] ALVES, L. Merger in Brazil Creates 5th Largest Stock Market in the World | The Rio Times | Brazil News. The Rio Times. Disponível em: <http://riotimesonline.com/brazil-news/rio-business/merger-in-brazil-creates-5th-largest-stock-market-in-the-world/>. Acessado: 01/08/2017.

[10] BOSCHETTI, A.; MASSARON, L.. *Python data science essentials*. Packt Publishing Ltd, 2016.

[11] TREVIZAN, K. Após ter pregão interrompido, Bovespa retoma atividade e opera em forte queda. G1. Available at: <http://g1.globo.com/economia/mercados/noticia/bovespa-180517.ghtml>. Acessado: 01/08/2017.

[12] NAU, R. Statistical forecasting: notes on regression and time series analysis. people.duke.edu. Dísponível em: <http://people.duke.edu/~rnau/411home.htm>. Acessado: 01/08/2017.

[13] MUSHTAQ, R. Augmented Dickey Fuller Test. 2011.

[14] BROWNLEE, J. How to Check if Time Series Data is Stationary with Python. Machine Learning Mastery. Disponível em: <http://machinelearningmastery.com/time-series-data-stationary-python/>. Acessado: 01/08/2017.

[15] Forecast a time series with ARIMA in Python. datascience.ibm.com. Disponível em: <https://datascience.ibm.com/exchange/public/entry/view/815137c868b916821dec777bdc23013c>. Acessado: 01/08/2017.

[16] Building ARIMA models. PennState Eberly College of Science. Disponível em: <https://onlinecourses.science.psu.edu/stat510/node/25>. Acessado: 01/08/2017.

[17] NIELSEN, M. A. Neural Networks and Deep Learning. Determination Press, 2015.

[18] OLAH, C. Understanding LSTM Networks. Disponível em: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>. Acessado: 01/08/2017.

[19] HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, v. 9, n. 8, p. 1735-1780, 1997.

[20] GUO,J. Backpropagation through time. Não publicado. **Harbin Institute of Technology**, 2013.

[21] Keras Documentation. keras.io. Disponível em: <https://keras.io/getting-started/faq/#what-does-sample-batch-epoch-mean>. Acessado: 01/08/2017.

[22] BABYAK, M. A. What you see may not be what you get: a brief, nontechnical introduction to overfitting in regression-type models. **Psychosomatic medicine**, v. 66, n. 3, p. 411-421, 2004.

[23] HEATON, J. Introduction to Neural Networks for Java. 2. ed. [s.l.]: Heaton Research, Inc, 2008.

[24] TUKEY, J. W. Exploratory data analysis. Reading, Mass. [u.a.]: Addison-Wesley, [s.d.].

[25] HALLS-MOORE, M. White Noise and Random Walks in Time Series Analysis. Quantstart.com. Disponível em: <https://www.quantstart.com/articles/White-Noise-and-Random-Walks-in-Time-Series-Analysis>. Acessado em: 01/08/2017.

[26] Repositório do Github com o trabalho realizado. Disponível em: <https://github.com/juliolrmonteiro/b3\_summer\_deeplearning/>. Acessado em 10/11/2017.