TRABALHO DE GRADUAÇÃO

PREVISÃO DE VALORES DE AÇÕES UTILIZANDO *DEEP LEARNING*

Pedro Henrique Hecksher Faber

Brasília, Julho de 2016

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

PREVISÃO DE VALORES DE AÇÕES UTILIZANDO *DEEP LEARNING*

Pedro Henrique Hecksher Faber

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Alexandre Romariz, ENE/UnB _____
Orientador

Prof. Alexandre Zaghetto, ENE/UnB _____
Examinador interno

Prof. Luís Figueredo, ENE/UnB _____
Examinador interno

Brasília, Julho de 2016

FICHA CATALOGRÁFICA

PEDRO HENRIQUE HECKSHER FABER
Previsão de valores de ações utilizando *deep-learning*,

[Distrito Federal] 2016.

69p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2016
Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. rede neural convolucional 2.perceptron multicamada
3. mercado financeiro 4.*deep learning*

I. Mecatrônica/FT/UnB II. Previsão de Valores de Ações Utilizando *Deep Learning*

REFERÊNCIA BIBLIOGRÁFICA

FABER, PEDRO HENRIQUE HECKSHER, (2016). Previsão de valores de ações utilizando *deep-learning*. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$07, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 69p.

CESSÃO DE DIREITOS

AUTOR: Pedro Henrique Hecksher Faber

TÍTULO DO TRABALHO DE GRADUAÇÃO: Previsão de valores de ações utilizando *deep-learning*.

GRAU: Engenheiro ANO: 2016

Pedro Henrique Hecksher Faber

SHIS QL 16, Conj. 1 , nº 2, Lago Sul.

71640-215 Brasília – DF – Brasil.

Dedicatória

Eu dedico esse trabalho à você, minha irmã. Em breve será a sua vez de fazer essa jornada e traçar os seus próprios caminhos. Que durante todo esse trajeto você seja sempre a pessoa brilhante que é hoje e que suas escolhas levem à realização de todos os seus sonhos mas, principalmente, que você seja sempre feliz.

Pedro Henrique Hecksher Faber

Agradecimentos

Agradeço aos meus colegas de curso, que dividiram cada noite virada, cada trabalho extenuante, cada tensão antes de prova. Agradeço à todos os membros do LARA, que me acolheram no laboratório de braços abertos e me ajudaram intensamente durante essa reta final. Agradeço à todos os meus amigos, dentro e fora do curso. Eles me deram motivo para sorrir e continuar mesmo nas integrais mais complexas, nos espaços de estados mais difíceis e nas transformadas Z que menos faziam sentido. Um agradecimento especial aos meus amigos Eric, De Hong, Daniel, Teixeira, Jessé, Caio, Carioca e Cris, cuja amizade e companheirismo foi essencial durante toda essa jornada. Agradeço à todos os professores espetaculares que tive durante o curso. Devo fazer uma menção especial ao Prof. Romariz, oráculo da inteligência artificial, pela orientação e ajuda durante todo o projeto.

Agradeço ao meu pai e a minha mãe por todo apoio e educação e pela formação da pessoa que sou hoje mas, principalmente, por todo amor que sempre me deram.

Pedro Henrique Hecksher Faber

RESUMO

O mercado financeiro é considerado um dos mais complexos e desafiantes sistemas criados pelo homem. Devido a grande quantidade de entradas e suas complexas interações, é difícil para um humano analisar e prever, com acurácia, valores futuros das ações. Entretanto, é possível aplicar sistemas inteligentes e modelos estatísticos para estimar esses valores futuros. Esse projeto utiliza duas diferentes redes neurais artificiais para fazer a predição de valores futuros das ações PETR4. As ANNs implementadas são uma rede perceptron multicamada e uma rede neural convolucional profunda. A entrada de ambas as redes é um banco de dados de interesse por tempo dos usuários da Google sobre temas selecionados e o histórico dos valores de mercado das ação selecionada. As séries temporais usadas como entrada foram segmentadas com 94% sendo dados de treinamento e 6% sendo dados de validação. A rede neural convolucional profunda foi capaz de prever se as ações iriam subir ou descer 59.81% dos casos nos dados de validação. Comparando com a DCNN, o perceptron multicamada obteve um resultado melhor. Ele previu 68.22% dos dados de validação corretamente. Como esperado, as duas técnicas obtiveram resultados melhores que um classificador ingênuo, que teve uma taxa de acerto de apenas 50.72%.

Palavras Chave: rede neural convolucional, perceptron multicamada, mercado financeiro, *deep learning*

ABSTRACT

The financial market is considered one of the most complex and challenging man made systems. Owing to its numerous input variables and complex interactions, it is difficult for man to analyze and predict, with accuracy, future stock values. It is, however, possible to apply intelligent systems and statistical models to estimate these future values. This project uses two different artificial neural networks for estimating the future value of PETR4 stocks. The artificial neural networks implemented were a multilayer perceptron and a deep convolutional neural network. The input for these systems is a database of interest over time of Google users on certain topics and the historical market values of the selected stock. The time series used as inputs where segmented with 94% as training data and 6% validation data. The convolutional neural network was able to predict whether the stock was going to increase or decrease in the next day for 59.81% of the validation data tested. Compared to the DCNN, the multilayer perceptron was able to achieve a better result, accurately predicting 68.22% of the validation dataset. As expected, both techniques obtained better results than a naïve classifier, which was able to predict only 50.72% of the future states.

Keywords: convolutional neural network, multilayer perceptron, stock market prediction, deep learning

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SÍMBOLOS

Símbolos Latinos

e	Erro na saída
d	Valor desejado de saída
x	Entrada de uma rede neural artificial
w	Peso de uma conexão sináptica entre neurônios
s	Soma das entradas multiplicadas pelos pesos
y	Saída de um neurônio
l	Camada de interesse
L	Camada de saída
$\mathbf{x}$	Vetor ou matriz da entrada
$\mathbf{w}$	Vetor ou matriz de pesos entre camadas
$\mathscr{E}$	Função de erro
$\mathscr{S}$	Entropia

Símbolos Gregos

η	Taxa de aprendizagem
σ	Função de ativação do neurônio
δ	Gradiente local
α	Constante de *momentum*

Siglas

ANN	Rede Neural Artificial - *Artificial Neural Network*
CNN	Rede Neural Convolucional - *Convolutional Neural Network*
EQM	Erro Quadrático Médio
DCNN	Rede Neural Convolucional Profunda - *Deep Convolutional Neural Network*
PETR4	Ação da Petrobras com Prioridade na Distribuição de Dividendos

Capítulo 1

Introdução

Fazer previsões no mercado financeiro é um problema de grande complexidade. Existem diversas variáveis envolvidas na determinação dos valores das ações. Nesse projeto propõe-se desenvolver duas redes neurais aplicadas a esse problema. Elas devem ser capazes de detectar informações relevantes em um conjunto de dados de entrada e prever estados futuros do mercado.

1.1 Contextualização

Eugene Fama, ganhador do Prêmio Nobel de Economia de 2013, publicou, em Maio de 1970, um artigo intitulado "Efficient Capital Markets: A Review of Theory and Empirical Work". Esse trabalho propôs um conceito que passou a ser amplamente utilizado no mercado financeiro, que é a hipótese do mercado eficiente (EMH) [2][3]. A EMH diz que o preço dos ativos refletem toda a informação disponível, isso implica em um agente não conseguir obter, consistentemente, retornos acima da média do mercado devido ao fato do preço dos ativos reagirem às notícias.

Como nem toda informação é evidente para todos, ineficiências no mercado acabam ocorrendo. Métodos computacionais podem correlacionar dados e extrair essas informações não triviais, que poderiam passar desapercebidas por um analista técnico. Obter essas informações implica em uma vantagem competitiva para quem as possuí pois, não estando disponível para todos o mercado, não haveria uma reação eficiente, acarretando, para o investidor detentor das mesmas, num ganho médio acima de zero.

Quando uma notícia significativa se torna pública, ela implica numa mudança de comportamento dos investidores. Essa mudança de comportamento é um fator humano que, por sua natureza, é caótico. Consequentemente, os valores das ações variam dependendo das especulações geradas em resposta à notícia. Na Figura 1.1, por exemplo, vê-se uma queda na cotação PETR3, da Petrobras, após outubro de 2014, queda correspondente ao escândalo do Petrolão e aos resultados da eleição. O mercado, portanto, tem variações caóticas e quase imprevisíveis. Análises convencionais tentam aproveitar ineficiências no mercado que independem ou se alinham com as

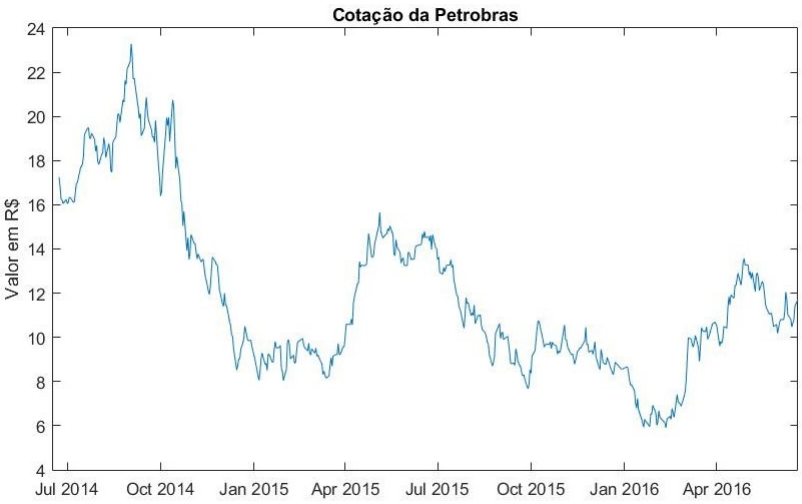

Figura 1.1: Gráfico da cotação da Petrobras nos últimos dois anos

especulações geradas por notícias.

Nos mercados atuais, técnicas convencionais, como análise de gráficos *candlesticks*, têm perdido espaço para técnicas mais inovadoras. Essa mudança tem acontecido devido ao aumento de investidores, que consequentemente, aumentam a velocidade de resposta do mercado a novas informações e tornam mais difícil encontrar ineficiências não vistas por outros investidores.

Quando há uma nova informação disponível no mercado, a procura por temas correlacionados feita em sítios de pesquisa como o Google aumenta, assim como o número de publicações sobre o assunto em redes sociais como Twitter e Facebook. A variação na popularidade do tema comumente antecede a resposta do mercado. Essa correlação entre procura e resposta é sutil e pode não ser simples de ser visualizada.

1.1.1 Tipos de Análise de Ativos

Nessa seção, as principais técnicas de análise de ativos serão abordadas.

1.1.1.1 Análise Fundamentalista

O primeiro método é a análise fundamentalista. Ele consiste em estudar a situação financeira da empresa, como balanço patrimonial, demonstração do resultado de exercício, fluxo de caixa, etc. Esses dados, apresentados em relatórios, são chamados de demonstrações financeiras.

Analistas fundamentalistas utilizam as demonstrações financeiras para estimar um valor de mercado para as empresas. Sabendo-se esse valor estimado, é possível estimar um valor para a ação da empresa sem especulação. Caso esse valor esteja mais baixo que o valor de mercado das ações, é um indicativo de que está num bom momento de vender, caso esteja acima do valor de mercado, significa que está em um bom momento de comprar. Entretanto esse não é o único

Figura 1.2: Exemplo de gráfico *candlestick*[0]

critério para a análise fundamentalista. Também se leva em conta a perspectiva de crescimento do mercado na área do ativo que está sendo comprado, no histórico da empresa, na sua credibilidade. A análise é feita em cima da empresa em si, normalmente para um investimento de médio a longo prazo.

A análise fundamentalista pode ser feita também em cima de commodities ou de moedas, baseando-se no mercado daquele produto ou da economia do próprio país.

1.1.1.2 Análise Técnica

A análise técnica pura segue a hipótese de que toda, ou quase toda a informação necessária para saber valores futuros de ativos está contida no próprio mercado. Analistas técnicos focam em determinar o valor futuro de ações baseando-se em uma análise numérica das tendências do mercado. Modelos matemáticos como média móvel, média móvel exponencial, bandas de Bollinger, e análise de Fourier são utilizados nesse tipo de análise.

Essa metodologia de especulação surgiu no século XVII com alguns conceitos primitivos e algumas máximas a serem seguidas [1]. No século seguinte, analistas japoneses desenvolveram o gráfico *candlestick* que contém as informações de abertura, fechamento (ou quotação). Esse gráfico contém informações de abertura, fechamento, máximo e mínimo em cada *candle*. Normalmente utiliza-se esse tipo de gráfico como base para outras análises técnicas. Na Figura 1.2, pode-se ver um gráfico *candlestick*, com duas médias móveis exponencias plotadas com períodos diferentes.

1.1.1.3 Análise Computacional

O terceiro método frequentemente utilizado originou-se da análise técnica, que é a utilização de métodos computacionais para auxiliar ou conduzir a análise. Atualmente, com a disseminação da computação, é incomum ver analistas técnicos trabalharem sem o auxílio de softwares e algoritmos computacionais.

Desde cálculos de médias móveis simples até aplicações de filtros de Kalman e *machine lear-*

[0]Gráfico extraído do site www.bussoladoinvestidor.com.br no dia 21/06/16

ning, a análise técnica foi revolucionada pelo advento da computação. *Algotraders* (analistas que desenvolvem algoritmos para comprar e vender ativos automaticamente) utilizam essas diversas técnicas para tentar ter alguma vantagem no mercado.

1.1.2 Técnicas de Aprendizado de Máquina

Aprendizado de máquina é uma área que trabalha com otimizações matemáticas com o objetivo de reconhecimento e detecção de padrões. Em termos gerais, técnicas de aprendizado de máquina tentam modelar um sistema ou prever resultados futuros com informações de saídas ou estados passados.

Existem diversos tipos de aprendizado de máquina, desenvolvidos para diferentes objetivos. Algumas redes surgiram através de inspirações na natureza e no funcionamento do cérebro humano, como algoritmos genéticos e redes neurais artificiais, outras vieram da generalização de modelos estatísticos, como as cadeias de Markov. Nessa seção será abordada, de forma sucinta algumas técnicas.

1.1.2.1 Redes Neurais Artificiais(ANN)

As redes neurais artificiais surgiram através da observação do sistema nervoso humano e como ele resolve problemas. Em redes artificiais, os "neurônios" são unidades de processamento com entradas e saídas se interconectando, dessa forma mimetizando sistemas nervosos.

Redes neurais artificiais possuem os seguintes pré-requisitos [4]:

- Unidades de processamento, chamadas de neurônios.
- Interconexões com diferentes pesos entre as unidades de processamento. Essas conexões definem como a ativação e a saída de um neurônio leva à entrada de um próximo.
- Regras de ativação que agem nos sinais de entrada para produzir uma saída.
- Padrões de aprendizado que ajustam os pesos dos entradas e saídas do sistema.

Os neurônios podem ser organizados de diversas formas mas, em geral, são organizados em camadas. Normalmente existe uma camada de entrada, em que os dados entram na rede, existe uma ou mais camadas ocultas, em que os dados são processados e uma camada de saída, onde os dados são manipulados para ficarem no seu formato de saída. A Figura 1.1.2.1 mostra uma representação desses três tipos de camadas. Os neurônios de h_1 à h_{k1} representam a primeira camada oculta, g_1 à g_{k2} representam a segunda camada oculta, e f_1 ef_2 representam a camada de saída. k_1 e k_2 são os números de neurônios da primeira e segunda camada escondida, respectivamente.

Como dito anteriormente, existem múltiplos tipos de ANNs, para diferentes propósitos. Redes *feed-forward* são as mais comuns, em especial os perceptrons e perceptrons multicamada*s*. Elas foram as primeiras a serem desenvolvidas e são de fácil implementação. Nesse tipo de rede o

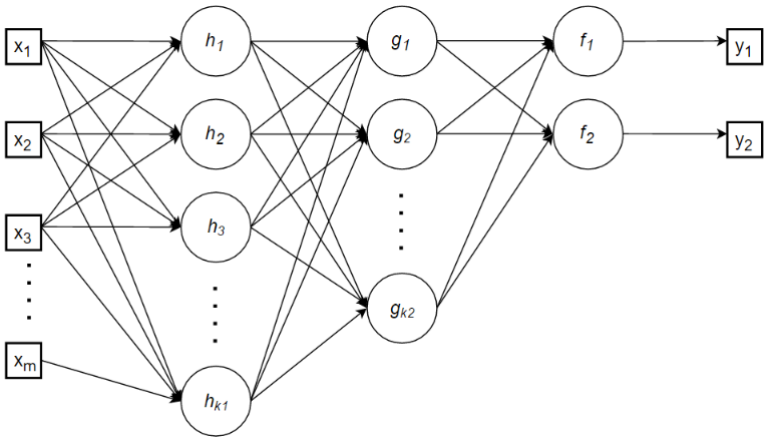

Figura 1.3: Exemplo de uma rede perceptron multicamada

movimento dos dados é unidirecional. Ele é recebido na entrada, passa pelas camadas escondidas e sai pela saída. Não existe um *feedback* interno na rede durante essa etapa.

O processo de aprendizado de todas as ANNs pode ser com dados durante a execução, de forma *online,* ou com dados captados anteriormente. A técnica mais comum de treinamento é chamada de *back-propagation.* Durante esse tipo de treinamento, para cada dado de entrada o computador calcula o erro entre a saída e o valor esperado real. Após o cálculo do erro, o sistema propaga esse erro para as camadas anteriores, ajustando os pesos entre as camadas. Seguindo esse processo, novos parâmetros são definidos para as entradas dos neurônios segundo uma taxa de aprendizado arbitrária, que pode ser tanto estática, quanto dinâmica, de forma a aproximar a saída do valor real esperado[2].

Repetindo esses passos para diferentes entradas no sistema, os pesos se ajustam de forma a tentar encontrar o menor erro de saída. Para uma base de dados extensa o erro tende a aproximar de um valore mínimo, local ou absoluto, para aquele banco de dados usado no treinamento. Caso o erro não chegue num valor mínimo, treina-se a rede múltiplas vezes em repetições chamadas de épocas.

1.1.2.2 *Deep Learning*

Deep learning é uma área da ciência da computação que utiliza algoritmos de aprendizagem de máquina para modelar sistemas que requerem um alto nível de abstração. Utilizando múltiplas camadas de processamento, normalmente se apresenta em forma de estruturas mais complexas que redes *perceptron* multicamada. Algoritmos de aprendizagem profunda têm sido usados em diversos campos e tem produzido alguns dos melhores resultados dentre as possíveis soluções com *machine learning*[5].

Redes de aprendizagem profunda, assim como redes neurais artificiais comuns podem ser utilizadas para as seguintes três classes de problema:

- Treinamento supervisionado: Neste tipo de aprendizagem existe um conjunto de dados de

treinamento que contém as saídas esperadas para diversas possíveis entradas, assim a rede varia seus parâmetros a fim de encontrar uma estrutura que encontre as saídas mais próximas dos valores esperados.

- Treinamento não supervisionado: Neste tipo de treinamento não existe (ou não é usado) um valor de saída desejado.
- Sistema híbrido: Possui parte do sistema sendo supervisionado e parte não, normalmente são mais complexos de serem implementados.

Dentre as técnicas mais implementadas de *deep learning*, vale mencionar uma: *convolutional neural networks*. Esse método teve um grande aumento em sua popularidade, em especial para processamento de imagens, devido à qualidade dos resultados que têm sido publicados com essa técnica. Para detecção de caracteres escritos a mão e leitura de texto, por exemplo, essa é a tecnologia *state-of-the-art* [6].

1.1.3 Trabalhos na Área

Durante a última década, vários pesquisadores e instituições tentaram prever valores de ativos no mercado utilizando redes neurais artificiais.

Uma relevante pesquisa foi conduzida na Nanyang Technological University [7]. A maioria dos estudos na área de previsão de valores de ações, tentam prever os valores um ciclo a frente, contudo, essa pesquisa tentou aplicar um algoritmo que fizesse uma previsão *multi-step* em que se tentava prever a situação do mercado vários estados a frente. Devido à complexidade do problema, os resultados foram fracos para previsões *multi-step*. Obtiveram, entretanto, resultados que os pesquisadores consideraram satisfatórios para predições imediatas, com uma taxa de acerto de 64.7% [7].

Outro resultado interessante foi obtido por um grupo de estudantes de doutorado de Harvard. Eles investiam no índice semanal S&P 500, seguindo as seguintes regras: Se houver um aumento previsto, eles investiriam todo o dinheiro em S&P 500; se houvesse um decréscimo previsto, eles removeriam todo o dinheiro investido e salvá-lo para mais tarde.

O diferencial desse projeto foi o uso de *big-data* para identificar os fatores com maior significância estatística no mercado. O grupo reuniu mais de 7.000 conjuntos de dados de assuntos relacionados a mercado financeiro Depois de usar um classificador Bayesiano a fim de eliminar os dados estatisticamente não significativos, o número de conjuntos de dados foi reduzido para 20. Aplicando estes 20 conjuntos como entradas em um perceptron multicamada, com um treinamento em *back-propagation* eles conseguiram uma precisão geral de 63,64%. Com uma precisão para aumentos de 79,37%, e para quedas de 36,11%. O treinamento incluiu dados de 1999-2010, e os utilizou os dados de 2011 e 2012 para teste. O modelo obteve, em média, um lucro 2.39% acima do crescimento do mercado mensalmente [1].

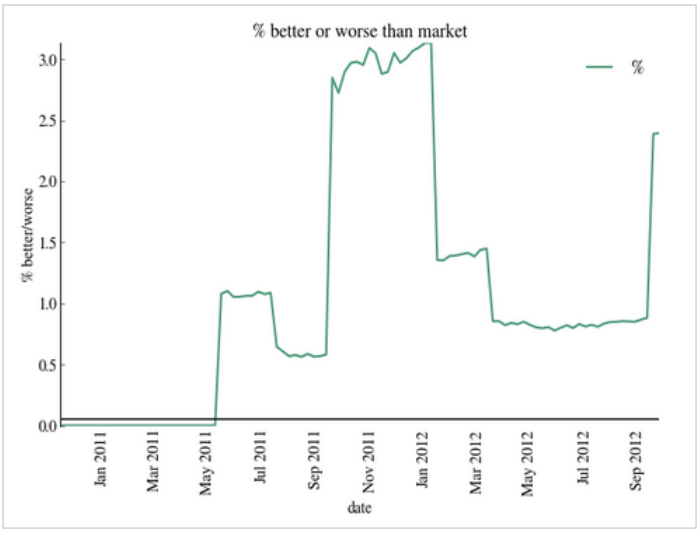

Figura 1.4: Ganho médio acima do mercado por mês [1]

Existiram, além dos mencionados, diversos outras pesquisas com bons resultados em prever valores no mercado financeiro [8, 9, 10, 11].

1.2 Definição do problema

O mercado financeiro é um sistema não linear de grande complexidade. Mesmo as modelagens mais robustas têm dificuldade de acertar previsões. Essa complexidade é intrínseca ao sistema, que depende de inúmeras variáveis. Para fazer previsões e acertar constantemente, um investidor teria que ter todas as informações envolvidas no mercado: todos os relatórios financeiros de cada empresa, mesmo os dados não publicados; quanto cada outro investidor vai investir e em qual ativo; os dados macroeconômicos de cada país, incluindo os não são divulgados ao público; além de diversos outros fatores como, fenômenos naturais e interações humanas. Como é, em termos práticos, impossível ter acesso a todas essas informações, a modelagem com o objetivo de fazer previsões e obter lucro, fica restrita às informações acessíveis.

1.3 Objetivos do projeto

Esse trabalho tem como objetivo desenvolver um sistema inteligente que seja capaz de prever se o preço de um ativo vai subir ou descer no dia seguinte. A rede neural artificial deve ser capaz de, sistematicamente, obter um lucro médio acima do mercado. O ativo selecionado para esse trabalho foi a ação PETR4, da Petrobras. O sistema proposto deve ter como entradas dados de *big data,* disponibilizados pelo Google Trends e dados de mercado, e deve ser capaz de utilizar esses dados para correlacionar popularidade, na plataforma de pesquisa Google, de temas pré-selecionados arbitrariamente com as ações da Petrobras.

1.4 Resultados obtidos

A taxa de acertos obtida para a rede neural convolucional foi de 59.81% para os dados de validação e 64.60% para os dados de treinamento. Para a rede perceptron multicamada desenvolvida a taxa de acerto foi de 68.22% para os dados de validação e 75.50% para os dados de treinamento. As duas redes obtiveram resultados significativamente superiores a um classificador ingênuo, que teve, para os dados de validação, uma taxa de acerto de 50.72%.

Executando simulações de investimento para o perceptron multicamada, em um período de 107 dias o rendimento médio é de 21.44%. A ação selecionada obteve um crescimento de apenas 1.85% para o mesmo período.

Capítulo 2

Fundamentos

"We've arranged a global civilization in which most crucial elements profoundly depend on science and technology." - Carl Sagan

Este capítulo contém as fundamentações teóricas necessárias para a compreensão do desenvolvimento do projeto.

2.1 Perceptron Simples

2.1.1 Estrutura do perceptron

O perceptron é o modelo mais básico de uma rede neural. A sua estrutura básica pode ser vista na Figura 2.1.

Seu funcionamento consiste em receber entradas, x_i, multiplicá-las pelos seus respectivos pesos, ω_i, e somar todos os resultados da multiplicação. Além das entradas do sistema, também soma-se um bias, que tem seu valor determinado como 1 e também é multiplicado por um peso respectivo, b, segundo a Equação 2.1.

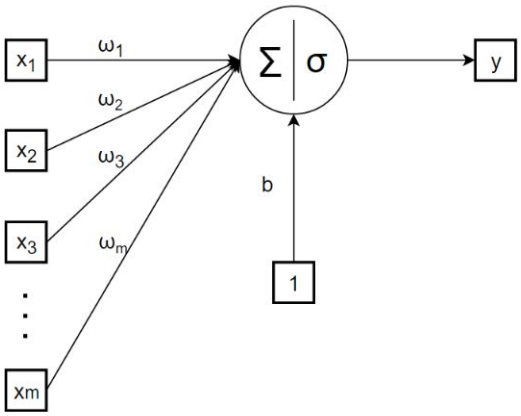

Figura 2.1: Perceptron simples

$$s = \sum_{i=1}^{m} \omega_i x_i + b \tag{2.1}$$

A soma das entradas ponderadas pelos seus pesos passa por uma função de ativação, σ, resultando na sua saída, y. As funções de ativação utilizadas nesse trabalho são as funções logística e tangente hiperbólica, dadas pelas Equações (2.2) e (2.3), respectivamente.

$$y = \sigma(s) = \frac{1}{1 + e^{-s}} \tag{2.2}$$

$$\sigma(s) = \tanh(s) \tag{2.3}$$

2.1.2 Convergência do perceptron

Suponha-se que, em um espaço $\mathbb{R}^3$ existam duas classes linearmente separáveis, como uma maça e uma tigela, dispostos lado a lado sobre uma mesa. Nesse cenário é fácil delimitar um plano que separe os dois objetos. Agora, assumindo as mesmas classes, ou objetos, só que com a maçã dentro da tigela, não é mais possível separá-las com um plano. É necessário uma superfície mais complexa. Esse exercício mental pode ser generalizado para qualquer espaço $\mathbb{R}^k$, sendo $k \in \mathbb{N}*$. Para esse espaço, podem existir duas classes, $\mathscr{C}_1$ e $\mathscr{C}_2$, que entre si são linearmente separáveis, possuindo um hiperplano que as separa, mas ao aproximá-las no espaço, elas perdem essa propriedade.

Um perceptron não é capaz de separar duas classes $\mathscr{C}_1$ e $\mathscr{C}_2$ caso elas não sejam linearmente separáveis. Portanto, para um perceptron simples, será assumido que $\mathscr{C}_1$ e $\mathscr{C}_2$ contém pelo menos um hiperplano que as separam em $\mathbb{R}^k$.

Sendo os dados de entrada da rede representados através de um vetor

$$\mathbf{x}(n) = \begin{bmatrix} 1, & x_1(n), & x_2(n), & x_3(n), & ..., & x_m(n) \end{bmatrix}^T \tag{2.4}$$

e, de forma correspondente, os pesos

$$\mathbf{w}(n) = \begin{bmatrix} b(n), & w_1(n), & w_2(n), & w_3(n), & ..., & w_m(n) \end{bmatrix}^T \tag{2.5}$$

Utilizando as Equações (2.4) e (2.5) pode-se representar a Equação (2.1) da seguinte forma

$$s = \mathbf{w}^T(n)\mathbf{x}(n) \tag{2.6}$$

O conjunto de dados que serão usados como entrada no perceptron são: $\mathscr{H}_1$ o subespaço que contém todos os vetores de treinamento $\mathbf{x}_1(1), \mathbf{x}_1(2), ...$, pertencentes às classe $\mathscr{C}_1$ e $\mathscr{H}_2$ o subespaço que contém todos os vetores $\mathbf{x}_2(1), \mathbf{x}_2(2), ...$, pertencentes à classe $\mathscr{C}_2$. $\mathscr{H}_1 \cup \mathscr{H}_2$ representa todo o espaço de entradas $\mathscr{H}$.

Para o espaço de entradas $\mathscr{H}$ o treinamento envolve ajustar os pesos $\mathbf{w}$ para que seja possível afirmar que

- $\mathbf{w}^T\mathbf{x} > 0$ para cada vetor de entrada $\mathbf{x}$ pertencente à classe $\mathscr{C}_1$
- $\mathbf{w}^T\mathbf{x} \leq 0$ para cada vetor de entrada $\mathbf{x}$ pertencente à classe $\mathscr{C}_2$

A prova do teorema que demonstra que o perceptron converge (veja o Apêndice I), demonstra que é possível encontrar uma solução $\mathbf{w}_0$ que separe completamente as classes $\mathscr{C}_1$ e $\mathscr{C}_2$.

Pela demonstração no Apêndice I, o algoritmo de adaptação de pesos para o perceptron simples é:

1. Se o membro n do conjunto de treinamento $\mathbf{x}(n)$ foi corretamente classificado pelo vetor de pesos $\mathbf{w}(n)$, nenhuma correção é feita:

$$\mathbf{w}(n+1) = \mathbf{w}(n) \tag{2.7}$$

2. Caso contrário, os pesos $\mathbf{w}(n)$ são ajustados de acordo com a seguinte regra

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n)x(n) \quad \text{se } \mathbf{w}^T\mathbf{x} > 0 \text{ e } \mathbf{x} \text{ pertencente à classe } \mathscr{C}_2 \tag{2.8}$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)x(n) \quad \text{se } \mathbf{w}^T\mathbf{x} \leq 0 \text{ e } \mathbf{x} \text{ pertencente à classe } \mathscr{C}_1 \tag{2.9}$$

Em que η é o fator de aprendizagem e está definido entre 0 e 1. O valor de η para esse resultado não é importante pois, mesmo alterando o número de iterações necessárias para separar as classes, ele não altera a separabilidade delas em si.

2.2 Perceptron Multicamada

Retornando à analogia da maçã e da tigela, para um espaço $\mathbb{R}^3$, foi demonstrado que o perceptron simples consegue separar os dois objetos, desde que eles sejam linearmente separáveis. O perceptron simples consegue delimitar um plano no qual a maçã está de um lado e a tigela está do outro. Em um cenário em que a maçã e a tigela estão em uma posição que nenhum plano consiga separar os dois, como quando a maçã está dentro da tigela, o perceptron falha em separar os dois. O perceptron multicamada, entretanto, é uma modificação do perceptrons simples que consegue fazer a separação de duas classes, ou objetos, que não são linearmente separáveis. Quando a maçã está dentro da tigela, por exemplo. Novamente essa analogia pode ser generalizada para um espaço $\mathbb{R}^k$, sendo $k \in \mathbb{N}*$. [12]

2.2.1 Estrutura do Perceptron Multicamada

O perceptron multicamada é uma estrutura ordenada, evidentemente em camadas, de perceptrons simples. A Figura 1.3 contém um exemplo de uma rede desse tipo, cada neurônio contido na Figura 1.3 é representado pela Figura 2.1.

O perceptron multicamada é segmentado em três tipos de camada diferentes:

1. Camada de Entrada: que é composta pelos dados de de entrada de cada iteração.
2. Camada Escondida: que são camadas cujas saídas são entradas de próximas camadas.
3. Camada de Saída: em que sua saída é a saída do sistema em si.

O perceptron multicamada necessariamente contém pelo menos uma camada de entrada e uma camada de saída. Pode haver qualquer número de camadas escondidas.

Os perceptrons que não estão na saída do sistema, ou seja, neurônios de camadas escondidas, têm a saída de suas funções de ativação como entrada de perceptrons da próxima camada. Essa estrutura, embora de fácil compreensão, contém uma dificuldade intrínseca: não existe uma forma de medir, diretamente, o erro de neurônios das camadas escondidas. Cada neurônio, é portanto projetado para executar duas tarefas:

1. Calcular o sinal de saída, que é expressado através de uma função não linear em relação ao sinal de entrada $\mathbf{x}(n)$ e aos pesos $\mathbf{w}(n)$, também chamados de pesos sinápticos.
2. Computar uma estimação do vetor gradiente do erro em respeito aos pesos da entrada do neurônio, que é necessário para a propagação do erro para trás.

2.2.2 Funcionamento do Perceptron Multicamada

Com essa estrutura definida, o funcionamento da rede, para uma iteração, ocorre da seguinte forma:

1. Inicialização: É o processo no qual se definem os parâmetros iniciais da rede
2. Apresentação dos dados de treinamento: A primeira camada da rede é alimentada com os sinais de entrada.
3. Feed-Forward: É processada a propagação dos sinais para frente, conhecida como *feed-forward*. Ela funciona assim como o perceptron simples, para o neurônio j na camada l a soma das entradas é

$$s_j^{(l)}(n) = \sum_i w_{ji}^{(l)}(n) y_i^{(l-1)} \tag{2.10}$$

aonde $y_i^{(l-1)}$ é a saída da camada anterior. Aplicando uma função de ativação σ a saída do neurônio fica

$$y_j^{(l)} = \sigma_j(s_j^{(l)}(n)) \tag{2.11}$$

e se o neurônio for da primeira camada escondida

$$y_j^{(1)} = \sigma_j(\sum_i w_{ji}^{(1)}(n) \mathbf{x}_i(n)) \tag{2.12}$$

Se o neurônio j pertence à camada de saída (i.e., $l = L$ e L é a profundidade da rede),

$$y_j^{(L)} = o_j \tag{2.13}$$

o sinal de erro computado na última camada é

$$e_j(n) = d_j(n) - o_j(n) \tag{2.14}$$

no qual $d_j(n)$ é o valor esperado para a saída do neurônio j na última camada.

4. Back-Propagation: É a propagação para trás do erro pelas camadas, computada os gradiente locais $\delta_j^{(l)}$

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}\sigma_j'(s_j^{(L)}(n)) & \text{para um neuronio } j \text{ na camada de saída } L \\ \sigma_j'(s_j^{(L)}(n))\sum_k \delta_k^{(l+1)}(n)w_{kj}^{(l+1)}(n) & \text{para um neurônio } j \text{ em uma camada escondida } l. \end{cases} \tag{2.15}$$

Na qual σ' denota a diferenciação da função de ativação. Após o cálculo do erro, os pesos são ajustados de acordo com a uma generalização da Regra Delta

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[\Delta w_{ji}^{(l)}(n-1)] + \eta\delta_j^{(l)}(n)y_i^{(l-1)}(n) \tag{2.16}$$

aonde η é a taxa de aprendizagem e α é a constante de *momentum.*

Os passos 3 e 4 são repetidos para novas épocas até que critérios predeterminados de parada sejam cumpridos.

A demonstração do algoritmo de *back-propagation* está no Apêndice II.

2.3 Rede Neural Convolucional

Mais uma vez voltando à maçã e a tigela, a rede neural convolucional (CNN) faz a separação desses dois objetos de uma forma diferente das redes de perceptrons simples e multicamadas. Esse tipo de rede é capaz, assim como o perceptron multicamada de classificar objetos não linearmente separáveis, mas ao invés de delimitar uma superfície que separa os dois objetos, a CNN utiliza de *features* das entradas para identificar a qual classe elas pertencem. No caso da maçã e da tigela, por exemplo, a rede aprenderia a identificar propriedades que as distanciam uma da outra, como cor, formato, posição no espaço, etc.

2.3.1 Estrutura Básica das Redes Neurais Convolucionais

As redes neurais convolucionais são divididas em camadas. Os dados que entram na rede passam por um processo de *feed-forward* e o erro é propagado para trás por *back-propagation*, assim como o perceptron multicamada. A estrutura das camadas e a forma como o *feed-forward* e o *back-propagation* acontecem, no entanto, é fundamentalmente diferente.

A estrutura básica da rede pode ser vista na imagem 2.2.

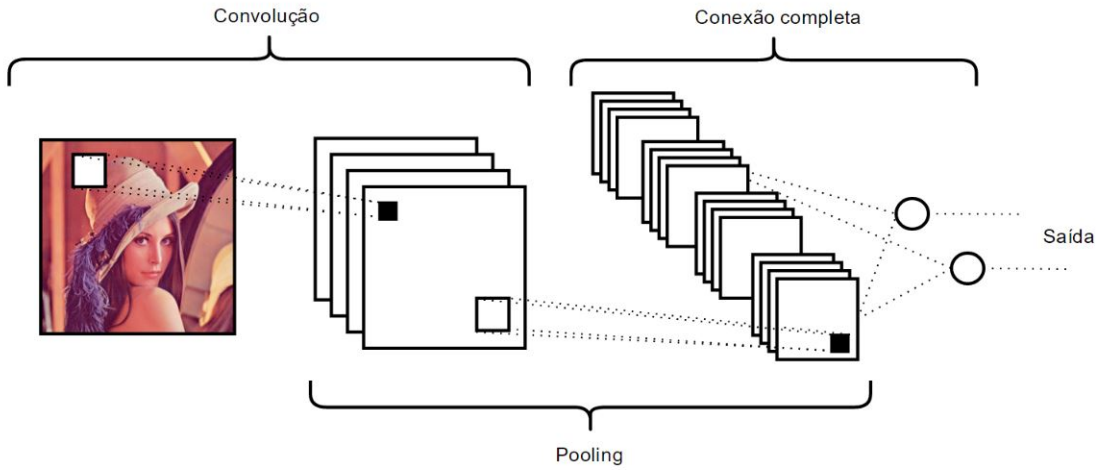

Figura 2.2: Exemplo de uma rede neural convolucional

Esse tipo de rede é dividido entre quatro tipos de camada:

1. Camada de Entrada: Assim como outras redes, essa é uma camada que contém os dados de entrada.

2. Camada de Convolução: A camada de convolução é composta por um número arbitrário de grades retangulares de neurônios, chamadas de *feature maps*. A entrada de cada neurônio de uma grade f_l específica recebe os dados de uma seção da malha anterior. Os pesos das entradas de cada neurônio são compartilhados entre todos os neurônios de uma mesma grade. Esses pesos só são diferentes entre os *feature maps*, cada um possuindo sua própria matriz de pesos. A camada de convolução tem esse nome porque, como cada neurônio de destino é uma composição de uma malha de neurônios anterior, ela é uma convolução espacial da sua entrada e os pesos são o filtro da convolução.

3. Camada de Pooling: A camada de *pooling* é mais simples que a camada de convolução. Nessa camada, seções retangulares da imagem são subamostradas de forma que saia apenas uma única saída desse processo. Essa subamostra comumente é feita a partir da seleção do valor máximo da seção.

4. Camada Completamente Conectada: Essa camada se apresenta no final da rede, normalmente após diversas camadas intercaladas de convolução e *pooling*. Essa camada é similar ao perceptron multicamada: todos as saídas de cada neurônio da camada anterior são conectados a perceptrons de saída.

2.3.2 Funcionamento da Rede Neural Convolucional

As redes neurais convolucionais normalmente são aplicadas para o processamento de imagens, como detecção de objeto e reconhecimento de padrões. Como a entrada é, consequentemente,

sempre uma matriz, os filtros de convolução as regiões de *pooling* normalmente são submatrizes quadradas, representados por tamanhos $m \times m$ e $k \times k$, respectivamente. Como no projeto foram utilizadas múltiplas séries temporais, as demonstrações a seguir serão feitas para tamanhos genéricos $m_1 \times m_2$ e $k_1 \times k_2$.

2.3.2.1 Propagação dos dados na camada convolucional

Suponha-se que exista uma camada de neurônios l, de tamanho $N_1 \times N_2 \forall (N_1, N_2) \in \mathbb{N}$, seguida de uma camada convolucional. Aplicando-se um filtro w de tamanho $m_1 \times m_2$ a camada convolucional vai ter um tamanho de $(N_1 - m_1 + 1) \times (N_2 - m_2 + 1)$. A soma das entradas de cada neurônio de l será, então

$$x_{ij}^{l} = \sum_{a=0}^{m_1-1} \sum_{b=0}^{m_2-1} w_{ab} y_{(i+a)(i+j)}^{l-1} \tag{2.17}$$

Com essa soma, pode-se calcular a função de ativação do neurônio

$$y_{ij}^{l} = \sigma(x_{ij}^{l}) \tag{2.18}$$

2.3.2.2 Propagação dos dados na camada de *pooling*

A camada de *pooling* é, em geral, simples. Existe mais de um tipo de camada de *pooling*, contudo, nessa fundamentação, só será abordado o *max-pooling*.

Para aplicar o *max-pooling*, basta seccionar a matriz ou vetor de entrada em regiões de tamanho $k_1 \times k_2$. Para cada uma desses segmentos de entrada, será selecionado o neurônio de maior valor para passar o valor para a próxima camada. Apenas um neurônio da seção influência a próxima camada a cada iteração. Com um *pooling* para cada seção $k_1 \times k_2$, a matriz é reduzida a um tamanho $\frac{N_1}{k_1} \times \frac{N_2}{k_2}$.

2.3.2.3 Propagação dos dados na camada completamente conectada

Essa camada, presente no final da CNN, é como um perceptron multicamada normal. Todos os dados da última camada, convolucional ou de *pooling,* são conectados à perceptrons que computam a saída do sistema.

2.3.2.4 *Backpropagation* na camada convolucional

Pelo que foi visto na fundamentação das outras ANNs nesse trabalho, para o ajuste dos pesos das conexões de cada camada, é necessário calcular o gradiente de cada peso.

Assumindo uma função de erro $\mathscr{E}$ para uma camada l de convolução, o cálculo do erro da camada $l-1$ se dá baseado na derivada parcial do custo $\mathscr{E}$ em relação a cada saída de cada

neurônio, $^{\partial \mathscr{E}}/_{\partial y^l_{ij}}$. Primeiro, calcula-se a o componente do gradiente de cada peso aplicando a regra da cadeia em cima da soma da entrada de todos os neurônios em que os pesos ocorrem. É necessário somar todos os casos x^l_{ij} em que w_{ab} ocorre, já que há um compartilhamento de peso.

$$\begin{aligned}\frac{\partial \mathscr{E}}{\partial w_{ab}} &= \sum_{i=0}^{N_1-m_1}\sum_{j=0}^{N_2-m_2}\frac{\partial \mathscr{E}}{\partial x^l_{ij}}\frac{\partial x^l_{ij}}{\partial w_{ab}} \\ &= \sum_{i=0}^{N_1-m_1}\sum_{j=0}^{N_2-m_2}\frac{\partial \mathscr{E}}{\partial x^l_{ij}} y^{l-1}_{(i+a)(j+b)}\end{aligned} \tag{2.19}$$

A substituição

$$\frac{\partial x^l_{ij}}{\partial w_{ab}} = y^{l-1}_{(i+a)(j+b)} \tag{2.20}$$

pode ser feita tirando a derivada parcial da Equação 2.17 em relação a w_{ab}. Expandindo a expressão $^{\partial \mathscr{E}}/_{\partial x^l_{ij}}$pela regra da cadeia,

$$\begin{aligned}\frac{\partial \mathscr{E}}{\partial x^l_{ij}} &= \frac{\partial \mathscr{E}}{\partial y^l_{ij}}\frac{\partial y^l_{ij}}{\partial x^l_{ij}} \\ &= \frac{\partial \mathscr{E}}{\partial y^l_{ij}}\frac{\partial}{\partial x^l_{ij}}(\sigma(x^l_{ij})) \\ &= \frac{\partial \mathscr{E}}{\partial y^l_{ij}}\sigma'(x^l_{ij})\end{aligned} \tag{2.21}$$

Como já se sabe o valor de $^{\partial E}/_{\partial y^l_{ij}}$, para calcular o gradiente da função de erro, $^{\partial \mathscr{E}}/_{\partial x^l_{ij}}$, basta calcular a derivada da função de ativação $\sigma'(x)$.

A propagação dos erros para trás pode ser mostrada através de mais um uso da regra da cadeia.

$$\frac{\partial \mathscr{E}}{\partial x^{l-1}_{ij}} = \sum_{a=0}^{m_1-1}\sum_{b=0}^{m_2-1}\frac{\partial \mathscr{E}}{\partial x^l_{(i-a)(j-b)}}\frac{\partial x^l_{(i-a)(i-b)}}{\partial y^{l-1}_{ij}} \tag{2.22}$$

A partir da Equação 2.17 pode-se inferir

$$w_{ab} = \frac{\partial x^l_{(i-a)(i-b)}}{\partial y^l_{ij}} \tag{2.23}$$

Utilizando esse resultado e substituindo em 2.22 chega-se no seguinte

$$\frac{\partial \mathscr{E}}{\partial x^{l-1}_{ij}} = \sum_{a=0}^{m_1-1}\sum_{b=0}^{m_2-1}\frac{\partial x^l_{(i-a)(i-b)}}{\partial y^{l-1}_{ij}} \tag{2.24}$$

Esse resultado, entretanto, só faz sentido para entradas que estão m_1 pontos de distância das lateral esquerda da matriz de entrada e m_2 pontos do topo dessa matriz. Para resolver esse problema, computacionalmente se acrescenta um *padding* de zeros nas entradas.

2.4 Uso de Entropia como Função de Erro

Comumente, a função de erro utilizada no treinamento de ANNs é

$$\mathscr{E}(n) = \frac{1}{2}\sum_{j\in C} e_j^2(n) \tag{2.25}$$

Entretanto, pode-se substituí-la pela função de entropia cruzada do sistema

$$\mathscr{S} = -\frac{1}{n}\sum_{x}[d\ln y + (1-d)\ln(1-y)] \tag{2.26}$$

com d sendo o valor desejado e y a saída do sistema. Essa escolha ocorre devido ao fato da propagação de erro na entropia cruzada não depender da derivada da função de ativação, ou seja, não ocorre uma diminuição da aprendizagem quando o gradiente tende a zero. Não é trivial a conclusão que essa fórmula pode ser utilizada como uma função de erro. São duas propriedades que levam a essa conclusão.

A primeira é que a função não é negativa, ou seja $\mathscr{S} \geq 0$, isso é evidente pois o resultado do somatório é sempre negativo já que ambos os logaritmos estão entre 0 e 1. A segunda propriedade é que o valor da entropia cruzada decresce monotonicamente de acordo com o decrescimento do módulo do erro.

Na Equação 2.26, substituindo y por $\sigma(s)$ e aplicando a regra da cadeia tem-se

$$\begin{aligned}\frac{\partial\mathscr{S}}{\partial w_j} &= -\frac{1}{n}\sum_{x}(\frac{d}{\sigma(s)} - \frac{(1-d)}{1-\sigma(s)})\frac{\partial\sigma}{\partial w_j} \\ &= -\frac{1}{n}\sum_{x}(\frac{d}{\sigma(s)} - \frac{(1-d)}{1-\sigma(s)})\sigma'(s)x_j\end{aligned} \tag{2.27}$$

que pode ser simplificada a

$$\frac{\partial\mathscr{S}}{\partial w_j} = \frac{1}{n}\sum_{x}\frac{\sigma'(s)x_j}{\sigma(s(1-\sigma(s))}(\sigma(s)-d) \tag{2.28}$$

Para uma função de ativação logística

$$\sigma(s) = \frac{1}{1+e^{-s}} \tag{2.29}$$

A derivada da equação é simples, como visto a seguir

$$\begin{aligned}\frac{d\sigma(s)}{ds} &= \frac{e^{-s}}{(1+e^{-s})^2} \\ &= (\frac{1+e^{-s}-1}{1+e^{-s}})(\frac{1}{1+e^{-s}}) \\ &= (1-\sigma(s))\sigma(s) \\ \sigma'(s) &= \sigma(s)(1-\sigma(s))\end{aligned} \tag{2.30}$$

Que, substituindo na Equação 2.28 encontra-se[13]

$$\frac{\partial \mathscr{S}}{\partial w_j} = \frac{1}{n} \sum_x x_j(\sigma(s) - d) \tag{2.31}$$

Capítulo 3

Desenvolvimento

"If a machine is expected to be infallible, it cannot also be intelligent." - Alan Turing

O projeto consiste em utilizar dados do mercado e popularidade de determinados tópicos no sistema de busca da Google Inc. para estimar aumentos ou quedas nos valores de ações da Petrobras S. A.

O trabalho realizado pode ser dividido em duas etapas principais: a primeira é a aquisição de dados da plataforma Google Trends e dados de ações da Petrobras e a segunda etapa é a de programação e aplicação dos dados na rede.

3.1 Aquisição de Dados

3.1.1 Aquisição de dados do Google Trends

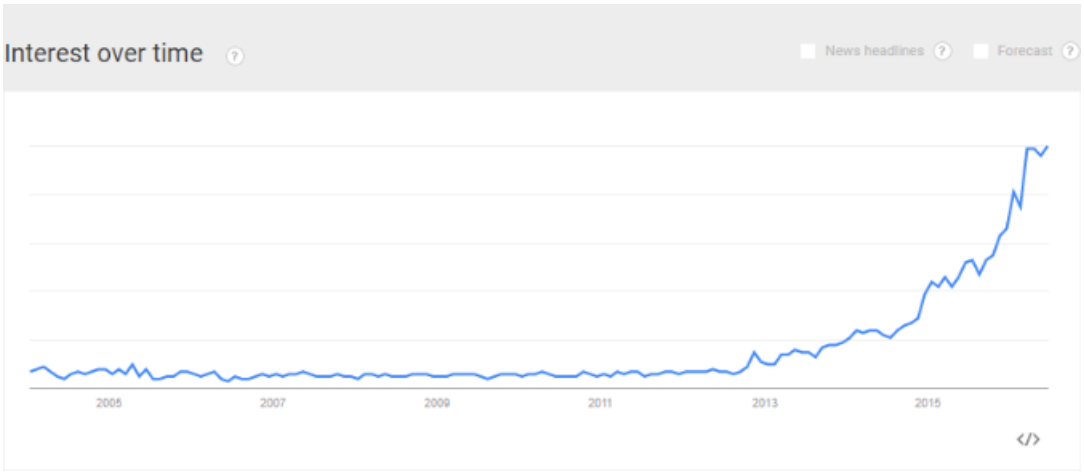

Figura 3.1: Gráfico de popularidade do termo *deep learning* nos últimos anos

O Google Trends é uma plataforma online fornecida pela Google Inc. que oferece dados sobre a popularidade de pesquisas no buscador da empresa. Os dados são oferecidos através de um gráfico, visível online, ou através de uma tabela, na extensão .csv disponível para download. Para obter

os dados, basta selecionar o período e pesquisar o tema de interesse. Um exemplo está disponível na Figura 3.1, aonde o tema pesquisado foi *Deep Learning* e o período foi de Janeiro de 2004 até Junho de 2016.

Os dados disponíveis contêm a informação normalizada em uma escala de zero a cem para o período de interesse selecionado, sendo sempre cem o dia de maior número de pesquisas, ou seja, os dados representam uma porcentagem em relação ao dia de maior valor.

Foram escolhidos arbitrariamente os temas de interesse, totalizando vinte:

- "ajuste fiscal"
- "corrupção"
- "diesel"
- "Dilma"
- "gasolina"
- "imposto"
- "índice Bovespa"
- "lava jato"
- "mercado financeiro"
- "Odebrecht"
- "OGX"
- "ouro"
- "PETR4"
- "Petrobras"
- "plataforma de petróleo"
- "pré sal"
- "PSDB"
- "PT"
- "Selic"
- "tesouro direto"

A escala temporal do gráfico depende do período selecionado, sendo possível ter uma precisão de até um minuto para tópicos populares em períodos recentes e curtos (menos de uma hora). Para a realização do trabalho, é de interesse dados fornecidos diariamente. Dados com um período diário só estão disponíveis caso o período selecionado seja de três meses ou menos.

Cada um contendo os dados diários normalizados de zero a cem desde Janeiro de 2011 até Maio de 2016. Existe, portanto, uma necessidade de capturar diversos períodos de dados, de três meses cada, para cada um dos temas e depois uni-los.

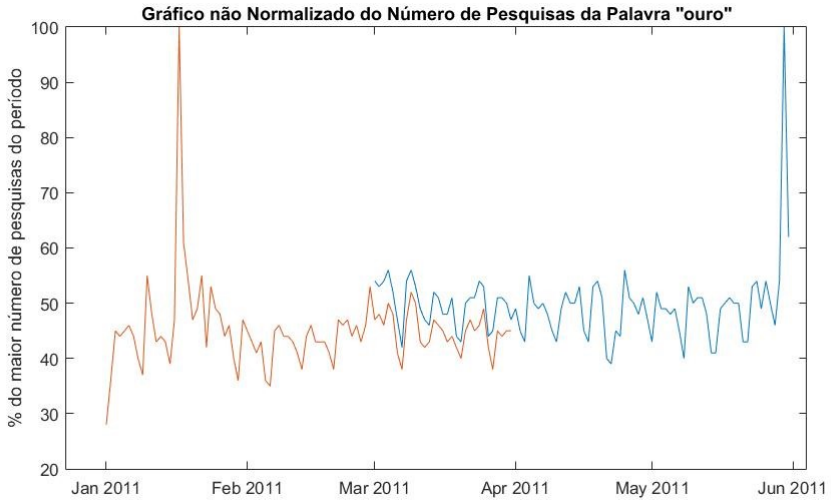

Figura 3.2: Gráfico exemplo de dados não ajustados

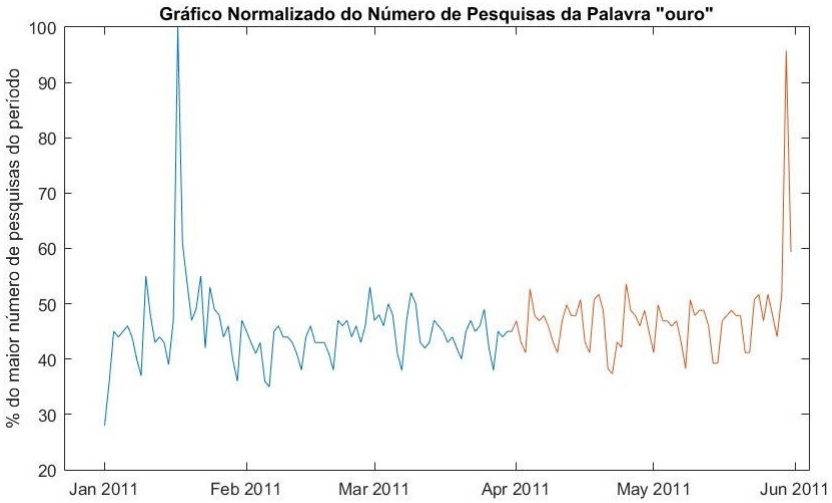

Figura 3.3: Gráfico exemplo de dados ajustados

Como cada conjunto de dados está normalizado para o período que ele representa, os dados foram capturados com uma interseção entre si no domínio do tempo. Assim, sabendo os valores finais de cada conjunto de dados e comparando-os com os primeiros dados do conjunto seguinte, é possível fazer uma nova normalização, novamente de zero à cem para o conjunto de dados

concatenados. Na Figura 3.2 pode-se ver um exemplo de um gráfico não ajustado. Nota-se que a parte laranja do gráfico está deslocada para cima em relação à azul. Após realizar o ajuste dos dados pode-se observar como fica o resultado na Figura 3.3.

Após esse pré-processamento dos dados, a série temporal final, para cada tópico, se inicia em Janeiro de 2011 e termina em Março de 2016. Para todo esse período, considerando as interseções e o limite máximo de três meses para cada aquisição de dados, são 31 arquivos no total, somando um total de 620 arquivos. Para capturar todos esses dados e processá-los, foram necessários dois *scripts* diferentes, um para capturar os dados do Google Trends, e outro para importar os arquivos para o software Matlab e automaticamente uni-los e normalizá-los.

3.1.2 Aquisição de dados da bolsa de valores

Além dos dados do Google Trends, para fazer estimativas em cima da bolsa de valores é importante ter o histórico daquela ação específica. Os dados utilizados foram os referentes à PETR4, que são ações da Petrobras S. A. com preferência na distribuição de dividendos. As informações contidas no histórico diário são os valores de abertura, os valores de fechamento, ou cotação, os máximos e mínimos, e o volume diário. Esses dados estão disponíveis em diversos sítios.

Para esse trabalho, foram utilizados dados extraídos da plataforma de dados econômicos da UOL, pois eles oferecem o mesmo banco de dados distribuído pela BMF&Bovespa, mas de forma organizada, em uma planilha para download. Os dados desse histórico, assim como os históricos finais e já pré-processados do Google Trends, começam em Janeiro de 2011 e vão até Março de 2016. Esses dados também precisam ser normalizados. Os valores de volume, por exemplo, são muito superiores a todos os outros valores.

Após a normalização, para implementação nos algoritmos de aprendizado de máquina com os dados do Google Trends, as séries da bolsa de valores não podem ter dados faltando. Os dados financeiros são disponibilizados, normalmente, como uma série temporal com dados faltando nas datas em que o pregão está fechado. Assim foi necessário um *script* que detecta datas faltando nos históricos, como finais de semana e feriados, insere essas datas nas séries, e insere o valor 0 para seus respectivos valores de mercado, visto na Figura 3.4. Posteriormente, durante o pré-processamento, esses valores iguais a 0 são substituídos por outros valores de interesse.

3.2 Implementação das Redes Neurais

Foram utilizadas duas técnicas de redes neurais artificiais para tentar estimar o se o valor das ações vão subir, ou se eles vão descer. A primeira técnica utilizada foi um perceptron multicamada, a segunda, uma rede neural convolucional. Para ambos os casos, as entradas das redes foram as séries temporais extraídas do Google Trends e os históricos de valores do mercado, somando um total de 25 séries temporais, cinco das quais são os dados do mercado e o resto do Google Trends. Por fim, além das redes, também foi utilizado um estimador ingênuo.

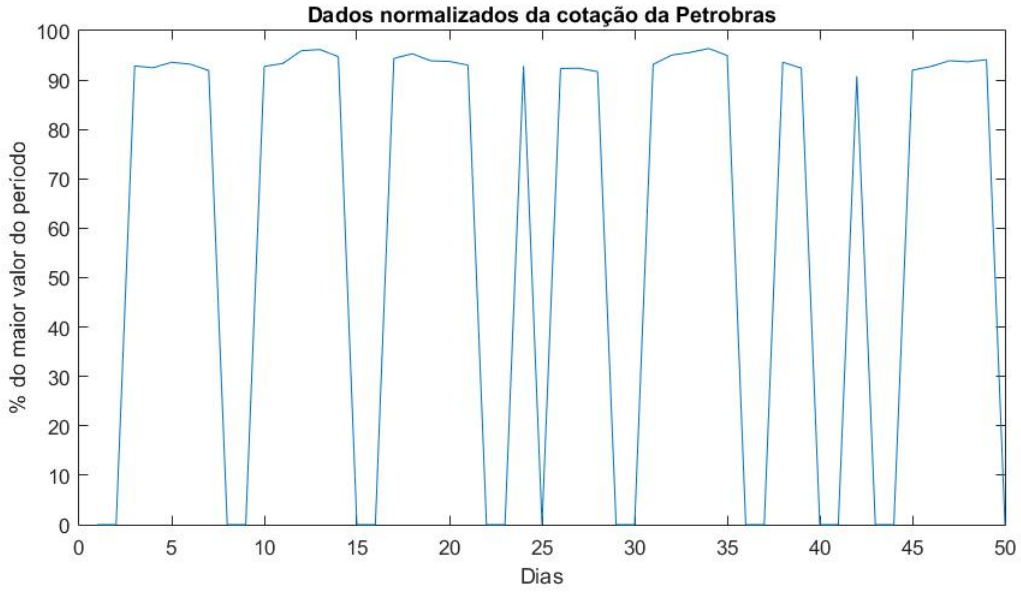

Figura 3.4: Exemplo de cotação após normalização e preenchimento com 0

3.2.1 Perceptron Multicamada

O perceptron multicamada implementado possui uma camada de entrada que contém os dados de estados anteriores da série multivariada. Possui duas camadas escondidas e uma camada de saída. Todas as camadas são completamente conectadas, ou seja, cada um de seus neurônios recebe dados de todos os neurônios da camada anterior e tem sua saída como entrada de todos os neurônios da próxima camada, com exceção da primeira e da última, que não têm camada anterior nem próxima, respectivamente.

Os dados de entrada, como dito anteriormente, são as séries temporais adquiridas. Para estimar o valor futuro, inserir todo o histórico prévio àquele valor, além de ser pesado computacionalmente, é impraticável. Necessitaria da inserção de dados na matriz de pesos a cada iteração já que o número de entradas aumenta. Isso implicaria nos dados serem tratados de forma irregular: os primeiros pesos seriam treinados durante mais iterações que os últimos. Foi decidido criar um período fixo, prévio ao estado que se quer estimar, em que todos os dados de todas as séries serviriam de entrada.

Esse período foi determinado através da correlação cruzada entre as séries do Google Trends e o valor das ações. A correlação cruzada representa a correlação entre duas séries discretas no tempo. Ela mede a similaridade entre uma série de interesse x (no caso o valor das ações da Petrobras) e copias de uma série y (os dados do Google Trends) deslocados no tempo. Pode-se testar a correlação deslocada para frente ou para trás no tempo, com um deslocamento máximo igual ao tamanho das próprias séries. Como as séries tem 1907 valores, testando todas as correlações possíveis, obtém-se 3814 valores de saída, para cada um dos -1907 à 1907 deslocamentos no tempo. O Apêndice III contém os gráficos das correlações cruzadas.

A determinação foi arbitrária após uma análise gráfica dos dados, não havia grande correlação entre o valor a ser estimado e valores mais distantes que 150 pontos, a frente ou atrás no tempo.

Considerando essa informação, foi selecionado um período de 150 dias para servir de entrada para a rede, para cada estimação. Para o dia 701, por exemplo, a entrada seria do dia 550 ao dia 700. Como são 25 séries, 150 dias de cada uma, para cada estimação entram no sistema, portanto, um total de 3750 valores de entrada.

Para essa quantidade de entradas foram encontradas duas principais dificuldades. A primeirA, e mais evidente, é a demora no processamento, que leva horas para cada centena de épocas treinadas. A segunda é o *overfitting*. A taxa de acerto nos dados de treinamento chega a valores próximos de 100%. Já nos arquivos de validação, o resultado ficava, com consistência, próximo de 50% de acerto, o que, para um sistema com saída binária, é o mesmo resultado esperado de uma decisão a partir de uma moeda não viciada. É um resultado insatisfatório.

Foram testados vários outros valores possíveis para esse tempo de entrada e foi notória uma melhora no resultado com a diminuição do número de dias de entrada. Seguindo essa tendência, os melhores resultados obtidos eram para uma entrada de apenas os últimos dois dias antes do dia de interesse.

Outro fator importante no tratamento da entrada é um pré-processamento dos dados. Todos os valores de entrada estão normalizados para um valor real entre zero e cem. Como o acesso da população à internet aumentou nos últimos anos, é esperado que a média das pesquisas dos temas tenha aumentado também. Consequentemente, os dados distribuídos de zero a cem contém uma informação que é difícil para a rede entender, pois a entrada para cada dado não representa todo o histórico, mas sim apenas os últimos dias. A entrada deve conter apenas a informação da variação de popularidade de um tópico, não a popularidade absoluta. Todos os dados são ajustados segundo as equações 3.1 e 3.2.

$$\Delta V(t) = V(t) - V(t-1) \tag{3.1}$$

$$\Delta V(0) = 0 \tag{3.2}$$

Aonde $V(t)$ é o valor de cada série na data t, $\Delta V(t)$ é o valor da variação na data. Os dados de entrada$V(t)$ são substituídos por $\Delta V(t)$.

O último ajuste na entrada foi tentar remover algumas das séries temporais e ver a resposta do número de acertos da saída do sistema. Percebeu-se que o resultado melhorava ao retirar as séries respectivas aos seguintes tópicos:

- "Diesel"
- "Corrupção"
- "Ajuste Fiscal"
- "Mercado Financeiro"
- "Índice Bovespa".

A rede utilizada possuí entrada, duas camadas escondidas e uma camada de saída. O primeiro parâmetro a se determinar é o número de neurônios na camada de saída. Esse valor deve corresponder à quantidade de dados de saída desejados para cada iteração.

Inicialmente, foram testados três neurônios de saída para a rede, uma para caso a cotação tivesse subido ou tivesse permanecido igual, uma saída para caso a cotação tenha diminuído em relação ao dia anterior e uma terceira saída, que deveria identificar quando os próximos valores de mercado seriam zero, que são os casos de finais de semana e feriados. O neurônio que tivesse o maior valor de saída é considerado ganhador daquela iteração, atribuindo o valor de 1 em sua respectiva posição em um vetor de três bits. Esse vetor era comparado com a *label* daquela iteração e, caso fossem iguais, era considerado um acerto.

Essa configuração inicial não resultou em uma boa taxa de acertos, para 50 neurônios em cada uma das camadas escondidas e com os dados de entrada já preprocessados, obteve taxas de acerto próximas de 33%, o que era esperado para uma saída aleatória em um sistema ternário. Consequentemente, essa configuração de saída foi descartada logo no início dos testes e foi substituída por uma saída binária, com dois neurônios. O primeiro para caso a cotação tenha subido ou se mantido estável, o segundo para caso a cotação tenha caído. Para ser possível ter uma saída com dois neurônios, os dados que foram definidos como zero foram substituídos pelo último valor não zero. Fazendo uma interpolação simples, como visto na Figura 3.5. Os valores iniciais das matrizes de peso de cada camada são randomizados segundo uma distribuição uniforme, entre -1 e 1.

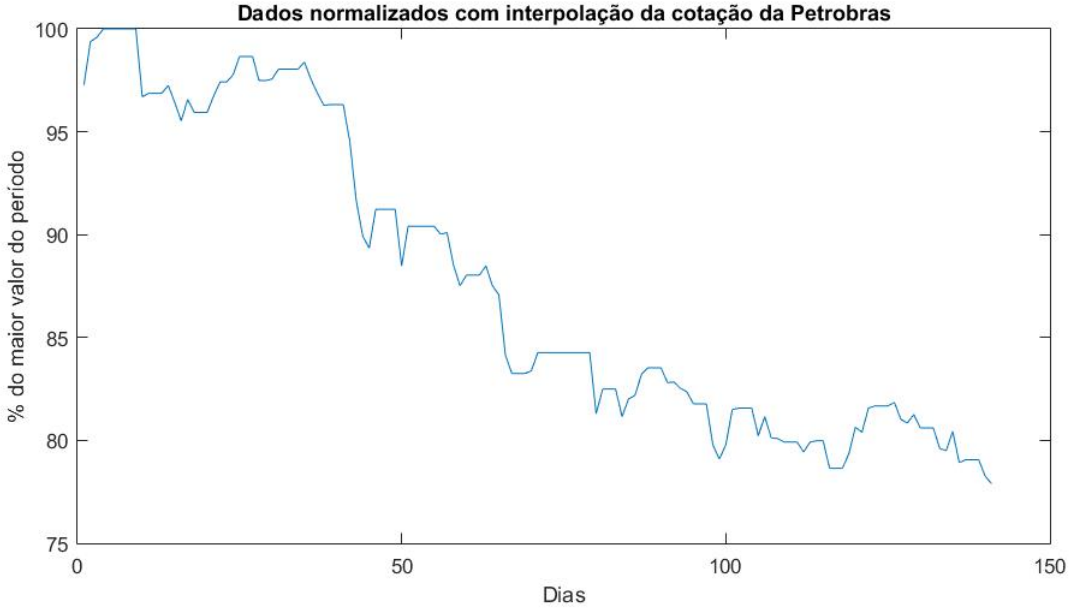

Figura 3.5: Cotação da Petrobras com interpolação durante um período de 5 meses

A definição dos outros parâmetros da rede se deu, em sua maioria, de forma empírica. Variaram-se os parâmetros até encontrar uma composição cujo resultado fosse satisfatório.

O número de neurônios nas camadas escondidas foi definido como 50. Foram testadas uma quantidade de neurônios de 20 à 100 variando de dez em dez, para cada uma das camadas in-

dividualmente, e o melhor resultado obtido foi para 50 neurônios em cada camada. A taxa de aprendizagem, η, também foi determinada empiricamente. Começando em um, foram testados diferentes valores para η de forma decrescente até encontrar um valor no qual há a queda mais rápida do erro quadrático médio sem que haja oscilação nos valores finais. As Figuras 3.6 e 3.7 representam quedas nos erros quadráticos médios para um η igual a 1 e 0.5, respectivamente.

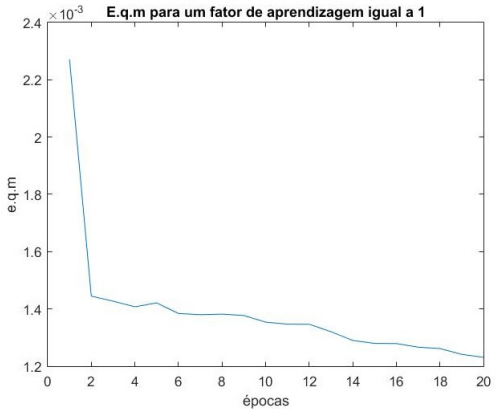

Figura 3.6: Erro quadrático médio nos dados de treinamento, por época, para um η igual a 1.

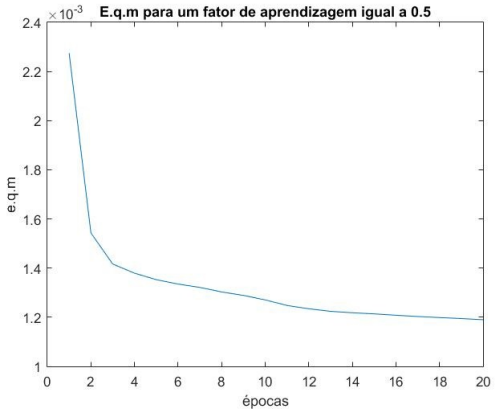

Figura 3.7: Erro quadrático médio nos dados de treinamento, por época, para um η igual a 0.5.

Para um η de 0.5, na Figura 3.7, são notórias a monotonicidade e a suavidade na queda do erro quadrático médio e, para η igual a 1, como o da Figura 3.6, essas propriedades não estão presentes. Os resultados se comportaram de acordo com o esperado em respeito ao η. Quanto maior, mais rápida a queda, contudo mais instável são os resultados; quanto menor, mais devagar é a queda, mas mais estáveis são os valores finais. Após diversas experimentações, foi perceptível que, para esse determinado sistema, qualquer η menor ou igual 0.5 já encontrava estabilidade em valores finais e resultava em taxas de acerto similares. O valor escolhido para η foi, consequentemente, 0.5.

Além de diferentes valores iniciais para η, também foi testado um η variável, com um valor em queda para cada iteração. Diferentes taxas de queda foram testadas para η, mas nenhuma resultou em diferenças estatisticamente significantes em relação a um η fixo.

Foram testados dois diferentes tipos de função de ativação: a função logística e a tangente hiperbólica. Nesse trabalho, a função logística teve resultados melhores que os da função tangente hiperbólica.

3.2.2 Rede Neural Convolucional

A CNN desenvolvida é segmentada em 10 camadas. A primeira é a camada de entrada que, como sempre, contém os dados de entrada do sistema. As próximas 4 camadas de convolução e 4 de *pooling*, dispostas de forma intercalada, começando por uma convolução seguida de *pooling*. Por fim, vem uma cada de saída completamente conectada.

Essa estrutura de rede, além de ter uma maior complexidade de implementação do que o perceptron multicamada, requer também mais capacidade computacional. Para treinamentos com o mesmo número de épocas e para o mesmo banco de dados, o tempo de execução do treinamento dessa rede é na ordem de 10^2 vezes o tempo do perceptron multicamada implementado. A Figura 3.8 mostra a estrutura da rede utilizada.

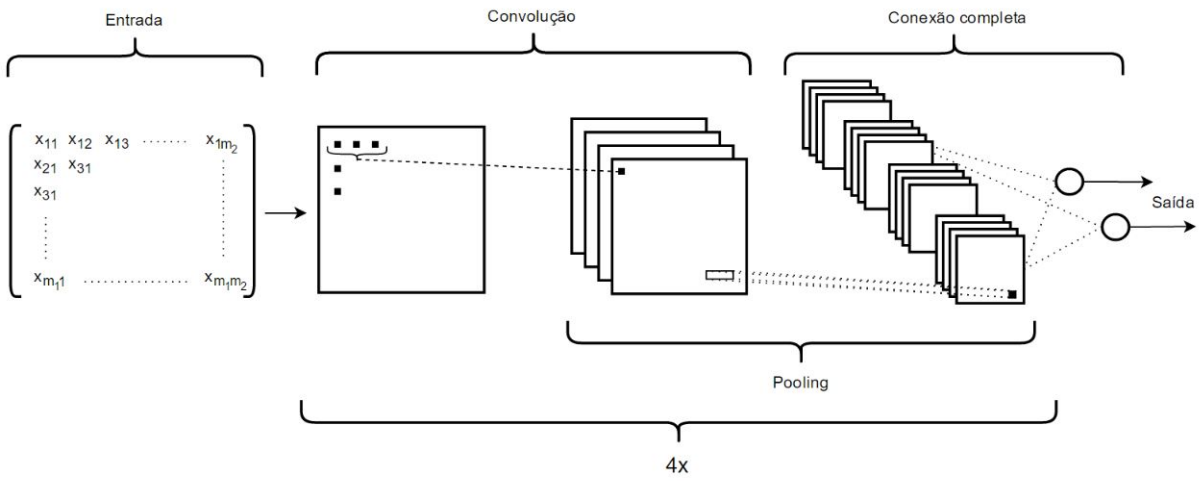

Figura 3.8: Disposição da CNN utilizada

Tratar séries temporais com CNNs é similar a problemas de classificação ou detecção de objetos em imagens. A diferença ocorre apenas na forma que os neurônios vão receber suas entradas. Uma imagem de entrada é representada através de uma matriz $N \times N$, da qual, na camada de convolução, um filtro $m \times m$ percorre a imagem fazendo a convolução de grupos retangulares de pixeis para a próxima camada. No caso de uma ou mais séries temporais, o filtro é um vetor de comprimento m e o *pooling*, ao invés de ocorrer em uma área retangular, ocorre só em cima de um trecho da série que está sendo amostrada.

A disposição dos vetores de entrada da CNN implementada está em formato matricial, sendo cada linha uma das séries de interesse. As razão para a estruturação dos dados dessa forma é a possibilidade de realizar operações matriciais com todo o conjunto de entradas simultaneamente. Embora aumente a complexidade do código, isso reduz, significativamente, o tempo de execução.

Assim como outras ANNs, a rede convolucional precisa de uma determinação prévia de suas condições iniciais. Como η, número de neurônios de saída, tamanho da janela de convolução e tamanho da janela de *pooling*.

As janelas de pesos usados como filtros na camada de convolução são gerados aleatoriamente no início de cada execução. A faixa na qual os pesos sinápticos são gerados foi definida arbitrariamente e, após alguns testes, foi definida entre-0.1 e 0.1.

As mesmas quantidades de neurônios de saída que foram testadas para o perceptron multicamada foram testadas para a camada completamente conectada da CNN, que também é a camada de saída. Contudo, nenhuma das duas quantidades de saída obtiveram resultados satisfatórios.

Inicialmente, foi decidido arbitrariamente que haveria três neurônios de saída, um para quando o valor do mercado subiu desde o último dia, um para quando o valor do mercado desceu desde o último dia e um para quando ficou estável ou é final de semana. Essa tentativa não rendeu bons resultados, a rede não aprendia a identificar quando era final de semana e ficava presa em mínimos locais óbvios. Quando, no período usado para treinamento, havia mais subidas do que descidas na cotação, a rede aprendia a sempre fazer a mesma estimativa: de que o valor iria subir. O mesmo acontecia para quando a maioria dos dados descia, a rede aprendia que o valor sempre iria descer.

A segunda tentativa foi com dois neurônios de saída, o número de neurônios que obteve os melhores resultados para o perceptron multicamada. Para esse número de neurônios, assim como no perceptron, foi necessário fazer uma interpolação dos dados de mercado dos finais de semana e feriados. Quando era final de semana ou feriado, o dado do último dia útil era repetido, implicando em ter uma variação diária igual a zero. Os treinos com duas saídas obtiveram o mesmo problema com mínimos locais que os treinos com três. Quando o valor das ações subia ou descia, na média, a rede ficava presa na estimativa de que o mercado sempre iria subir, ou descer, correspondentemente.

Para resolver esse problema de mínimos locais, foram testadas três abordagens. A utilização de *momentum* foi a primeira delas. Com a inserção desse parâmetro, o ajuste dos pesos não depende só do erro daquela iteração, mas também de uma porcentagem do ajuste da última iteração. Essa porcentagem é determinada por um valor α, como pode ser visto na equação 2.16. Dessa forma, se o gradiente descer para um mínimo local, os ajustes dos pesos teriam uma "inércia" que manteria a variação naquela direção. Diversos valores de α foram testados e resultaram em uma aprendizagem mais rápida, mas continuaram com o mesmo problema de mínimos locais.

A segunda abordagem foi a inserção de ruído gaussiano nos pesos a cada iteração. Essa foi a abordagem com piores resultados, a rede não ficava presa em mínimos locais, mas em contra partida não aprendia nada. Mesmo a inserção de pequenos ruídos com uma ordem de grandeza estimada de 10^{-3} em relação aos pesos da rede, não havia queda perceptível no gradiente.

A última abordagem testada foi a inserção de ruído branco gaussiano nos vetores de entrada. Inicialmente essa tentativa não teve um resultado satisfatório. Entretanto, mantendo esse ruído na entrada, após ajustes descritos futuramente, a rede foi capaz de aprender sem ficar presa em mínimos locais óbvios.

O último ajuste no número de neurônios foi a redução para apenas um. Essa saída implementada é diferente de saídas com múltiplos neurônios. Ao invés de ter uma saída respectiva a subida e uma respectiva a decida no valor da cotação, com apenas uma saída é necessário determinar um limiar arbitrário no seu valor. Para qualquer valor de saída acima desse limiar, a estimação é de subida, para qualquer valor abaixo desse limiar, a estimação é de descida.

Para a rede convolucional, mesmo com a inserção de ruído e com apenas uma saída, não foi possível observar bons resultados utilizando a função de erro EQM (Eq. 2.25). Essa função foi substituída pela entropia cruzada (Eq. 2.26) e obteve resultados significativamente melhores.

Durante todo o processo de ajuste de parâmetros, diversos valores para η foram testados. Os melhores resultados foram para $\eta = 0.001$. Como cada neurônio pré convolução influencia um

conjunto grande de neurônios da camada seguinte, a soma do erro propagado para trás tende a ser grande em relação aos pesos sinápticos. Portanto, para conter variações grandes nesses pesos, é necessário um η pequeno.

Capítulo 4

Resultados

"The most exciting phrase to hear in science, the one that heralds the most discoveries, is not "Eureka!" (I found it!) but 'That's funny..." - Isaac Asimov

Nesse capitulo serão apresentados e descritos os resultados dos treinamentos realizados. Os dados foram divididos com os primeiros 94% sendo treinamento e os últimos 6% sendo validação. Essa decisão foi tomada devido ao fato de que alguns termos extraídos do Google Trends só se tornaram populares e significantes para os valores das ações da Petrobras no final de no final de 2015 e início de 2016.

Antes de 2014 a correlação entre a popularidade do tema "lava jato" e o valor da ação PETR4 era insignificante, por exemplo. No final de 2015 e início de 2016 a correlação entre esses temas aumentou. Caso a segmentação entre treinamento e validação reservasse um período maior para validação, não haveria período algum nos dados de treinamento que a correlação entre "lava jato" e valor da PETR4 fosse estatisticamente significante. Nesse cenário, os dados do tema "lava jato" não seriam importantes para o treinamento, pois a rede iria perceber que não há nenhuma correlação entre os dois. Também não seriam importantes para a validação pois a rede aprendeu que esses dados não são importantes para o mercado, o que não é verdade.

4.1 Perceptron Multicamada

Ao contrário do esperado, o perceptron multicamada foi a técnica implementada que obteve melhores resultados. A execução do perceptron também é mais rápida que a execução da CNN. O banco de dados utilizado nos treinamentos e validações são as séries temporais dos temas citados em na Seção 3.1.1 e os dados de mercado da Petrobras.

Durante as 10 primeiras épocas de execução, a taxa média de acerto dos dados de validação foi de 61.19%, a taxa de acerto médio no treinamento foi de 69.43% e o tempo médio de execução foi de 11.10 s. Para as 10 próximas épocas, a taxa de acerto dos dados de validação média aumentou para 65.52%, a taxa de acerto médio no treinamento foi de 73.81% e o tempo médio de execução foi de 22.18 s. Nas vinte próximas épocas, chegando a 40 épocas de execução, a taxa de acerto dos dados de validação média foi de 62.91%, a taxa de acerto médio no treinamento foi para 84.52%, o tempo continuou aumentando linearmente em respeito ao número de épocas executadas. Ao totalizar 80 épocas, a taxa de acerto dos dados de validação caiu ainda mais, chegando a média foi de 62.24%, e a taxa de acerto médio no treinamento foi de 93.87% e o tempo final médio de execução foi de 88.10s.

A Figura 4.1 é gráfico do erro quadrático médio (EQM) por época referente a execução das primeiras 80 épocas.

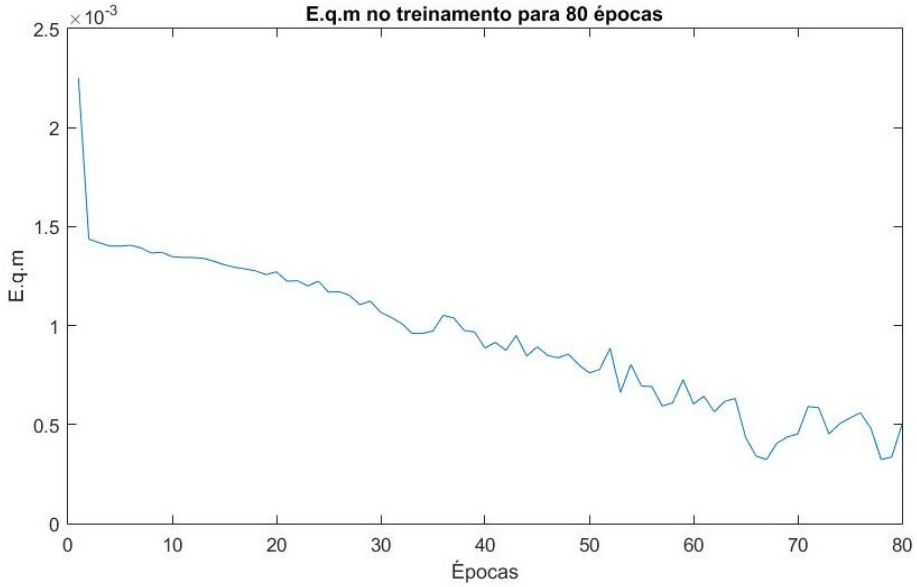

Figura 4.1: Gráfico da queda do EQM para 80 épocas

4.1.1 Discussão Sobre os Resultados do Perceptron Multicamada

Os resultados obtidos para o perceptron multicamada foram melhores do que o esperado. Os melhores treinamentos obtiveram 68.22% de acerto nos dados de validação.

Com o aumento no número de épocas, aumenta também a taxa de acerto nos dados de treinamento, monotonicamente. Esse resultado é esperado. É com base no banco de dados de treinamento que ocorrem os ajustes na rede.

Treinando a rede por 80 épocas, como visto na Figura 4.1, obtém-se uma taxa de acerto média de quase 94%. A taxa de acerto média para um treinamento de 20 épocas é de apenas 73.81%. Essa diferença, somada com uma queda na quantidade média de acertos nos dados de validação para esse período de épocas, categoriza um caso de *overfitting*. O número de acertos para os dados

de treinamento aumenta com o número de épocas. Entretanto, o número de acertos da validação aumenta apenas no começo e após algumas épocas, começa a cair devido ao *overfitting* do sistema.

Usando os parâmetros selecionados durante o desenvolvimento, os melhores resultados de validação para a rede perceptron multicamada ocorrem, normalmente, por volta de 20 épocas. Números de épocas menores que 20 normalmente resultam em treinamentos *underfitted,* mostrando que a rede não aprendeu tudo o que poderia para fazer estimações sobre os dados de validação. Treinamentos com dezenas de épocas a mais que 20 implicavam em *overffiting.* O treinamento ajustava os pesos tão bem para o período de treinamento do mercado que as previsões só funcionavam para ele.

Um parâmetro que influencia significativamente no resultado e vale ser mencionado é o número de dias passados que eram usados como entrada. A estimativa inicial de 150 dados não funcionou pois o impacto que as notícias têm no valor da ação provavelmente não geram resíduos que durem todo esse período. Embora exista a correlação entre dos dados, como visto no Apêndice III, a correlação mais forte é nos dias imediatos após a publicação da notícia. Um período longo de 150 dias contém, junto com informações importantes para o mercado, informações que são apenas ruído para a rede, dificultando o aprendizado. O melhor resultado obtido foi para um número de dias anteriores de entrada igual a 2.

4.2 Rede Neural Convolucional

Era esperado que a CNN implementada obtivesse resultados melhores que o perceptron multicamada. É um tipo de rede que tem obtido resultados estado da arte em trabalhos envolvendo processamento de imagens, por exemplo[14]. A razão do uso dessa técnica era a expectativa de que, assim como a rede funciona para detecção de *features* em imagens, ela fosse capaz de detectar *features* nas séries temporais e correlacioná-los. Embora tenham obtidos ótimos resultados, redes neurais convolucionais, por serem profundas e terem muitas iterações na propagação dos dados em cada camada, têm uma complexidade computacional maior que perceptrons multicamada. A rede implementada, para um treinamento de uma época com todo o banco de dados, demora consistentemente mais de 7 minutos, esse tempo de execução é centenas de vezes o tempo de treinamento de uma época do perceptron multicamada.

A CNN, ao contrário do que era esperado, teve um resultado pior do que o perceptron. A rede convolucional desenvolvida obteve, para um treinamento com 200 épocas, uma taxa de acerto para os dados de validação de 59.81% e, para os dados de treinamento, de 64.60%. Além de ter um resultado pior, o tempo de execução, como dito anteriormente, é mais extenso. O treinamento para 200 épocas demorou dois dias e, como pode ser visto no gráfico da Figura 4.2, o erro quadrático médio ainda estava em queda. É possível para para um treinamento mais longo, com mais épocas, a taxa de acerto da CNN aumentasse, em troca de mais tempo de execução.

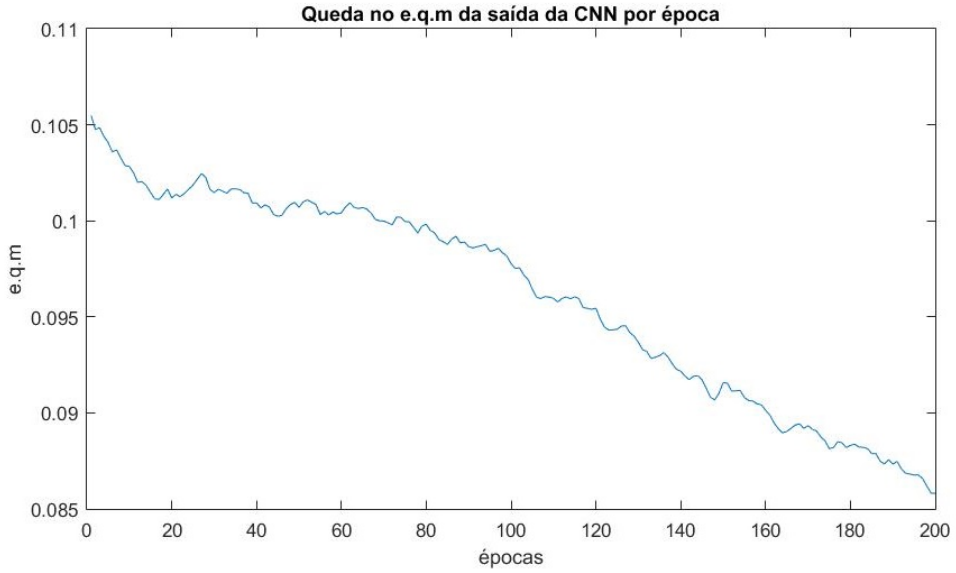

Figura 4.2: Queda no EQM por época de treinamento da CNN

Além de uma simulação com 200 épocas, também foi realizada uma simulação com 500. A execução para 500 épocas durou 4 dias. Entretanto, não houve estabilização do erro quadrático médio. Ao concluir o treinamento, o erro ainda estava em queda. Para essa quantidade de épocas treinadas, a taxa de acerto para o arquivo de validação foi a mesma que para 200 épocas, 59.81%.

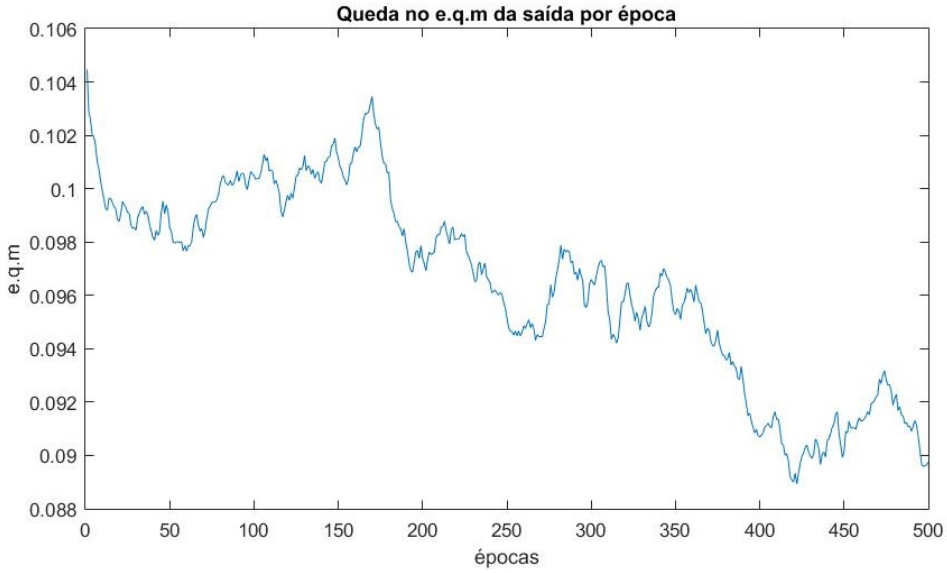

Figura 4.3: Queda no EQM por época de treinamento da CNN

4.3 Comparação dos Resultados

Para analisar o resultado das redes neurais artificiais implementadas, além dessas duas técnicas foi testado um classificador ingênuo nos dados de validação. Esse simples classificador estima o

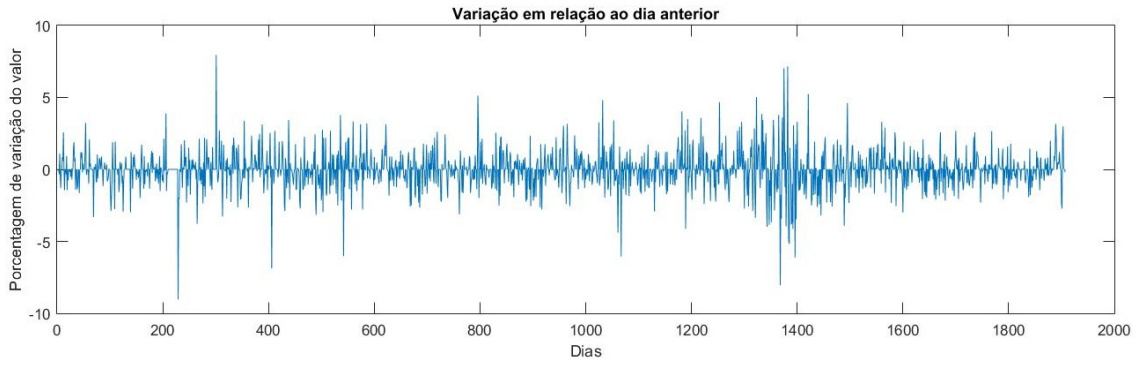

Figura 4.4: Variação percentual do valor da ação PETR4 em relação ao dia anterior para todo o banco de dados

estado de um dia baseando-se no dia anterior a ele, caso o valor da ação tenha subido, ele estimará uma subida, caso tenha descido, ele estimará uma descida. A gráfico da Figura 4.4 mostra a variação diária do valor da ação. Todo dia que a o valor no gráfico for positivo, significa que o valor da ação aumentou, toda vez que for negativo, significa que o valor da ação diminuiu. Através desses dados é fácil ver o resultado do classificador ingênuo.

Os dados de validação são os últimos 107 dias da série temporal vista na Figura 4.4. O classificador ingênuo teve uma taxa de acerto de 50.72% para esses dados. Ambas as redes neurais artificiais implementadas foram capazes de aprender com o conjunto de dados de entrada e encontrar um resultado melhor que o classificador ingênuo. A melhor taxa de acerto do perceptron multicamada foi de aproximadamente, 68.22%. O resultado obtido pela CNN foi de uma taxa de acerto para o arquivo de validação igual a 59.81%. É notório que as duas redes, para esse período, não só aprenderam com os dados fornecidos como também obtiveram resultados acima de um classificador não inteligente.

Comparando os resultados do perceptron multicamada e da CNN é evidente que o perceptron multicamada obteve resultados melhores. Essa afirmativa, no entanto, não pode ser generalizada. Ela só pode ser feita para essas redes, com seus parâmetros específicos e com esse banco de dados.

Como, para esse problema específico, o perceptron multicamada obteve resultados melhores, foram realizadas simulações de investimento utilizando ele como critério. As simulações executadas seguiam as estimativas do perceptron para o período de validação de acordo com as seguintes regras:

1. Inicia-se a simulação com R$ 1000.
2. Caso a rede preveja que o valor vai subir toda a quantia deve ser investida, caso ela já esteja aplicada, a aplicação deve ser mantida.
3. Caso a rede preveja que vai haver uma queda, toda a quantia deve ser desaplicada, caso ela já esteja, nada deve ser feito.
4. Caso a rede acerte uma subida, soma-se ao valor total a porcentagem que subiu.
5. Caso a rede acerte uma descida, nada deve ser feito ao valor total.

6. Caso a rede erre estime uma descida mas houve uma subida, nada deve ser feito ao valor total.

7. Caso a rede erre estime uma subida mas houve uma descida, subtrai-se do valor total a porcentagem da queda.

Essa simulação foi realizada 10 vezes e obteve, na média, um lucro de 21.44% em cima dos 107 dias usados como validação. Nas 10 simulações realizadas, o pior rendimento foi de 12.44% e o melhor, de 31.35%. Nesse mesmo período, a ação PETR4 valorizou 1.85%.

Capítulo 5

Conclusões

"Mathematics has given economics rigor, but alas, also mortis." - Robert Heilbroner

Este trabalho apresentou a aplicação de duas técnicas de redes neurais artificiais para a previsão em mercados financeiros. Foi explicado o processo de seleção de dados de entrada no sistema, assim como uma breve contextualização do problema. Apresentou a estrutura básica das redes implementas, bem como uma breve fundamentação matemática. Foram abordados o desenvolvimento das duas redes neurais artificiais aplicadas e as o processo de seleção de parâmetros. Por fim foram apresentados e comparados os resultados das estimações.

A rede perceptron multicamada, um sistema inteligente capaz de fazer estimações não lineares foi a primeira rede a ser implementada. Ela foi capaz de encontrar resultados satisfatórios e, em desacordo com as expectativas, melhores que os resultados da CNN implementada. A melhor taxa de acerto obtida foi de 68.22% para dados de validação.

A outra rede desenvolvida foi uma rede neural convolucional profunda, mais complexa e mais pesada computacionalmente que a outra técnica aplicada. Esperava-se que o resultado dessa ANN fosse melhor que o perceptron multicamada. Entretanto o resultado ficou aquém do esperado, acertando apenas 59.81% dos arquivos de validação.

As duas redes implementadas obtiveram resultados acima de um classificador ingênuo, que acertou apenas 50.72% das estimações durante o período de validação. Foram realizadas simulações de investimentos para o perceptron multicamada. Essas simulações obtiveram um lucro médio, em um período de 107 dias, de 21.44%.

Com trabalhos futuros propõe-se aplicar outras técnicas de sistemas inteligente para previsão de mercado. Também propõe-se utilizar um banco de dados mais vasto e com dados mais diversos, como dados direto de notícias. Outro objetivo futuro é de implementar esse algoritmo para operações de *day-trade* e não só operações diárias.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] SUN, J. R. J. S. K. A neural network application: Predicting the stock market. 2014.

[2] RUSSELL, I. *The Delta Rule*. [S.l.: s.n.], 1996.

[3] SNOWBERG, E.; WOLFERS, J.; ZITZEWITZ, E. *Prediction markets for economic forecasting*. [S.l.], 2012.

[4] RUSSELL, I. *Neural Networks Module*. [S.l.: s.n.], 1996.

[5] DENG, L.; YU, D. Deep learning. *Signal Processing*, Citeseer, v. 7, p. 3–4, 2014.

[6] CIRESAN, D. C. et al. Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, MIT Press, v. 22, n. 12, p. 3207–3220, 2010.

[7] DONG, G.; FATALIYEV, K.; WANG, L. One-step and multi-step ahead stock prediction using backpropagation neural networks. In: IEEE. *Information, Communications and Signal Processing (ICICS) 2013 9th International Conference on*. [S.l.], 2013. p. 1–5.

[8] PHUA, P. K. H.; MING, D.; LIN, W. Neural network with genetic algorithms for stocks prediction. In: *Fifth Conference of the Association of Asian-Pacific Operations Research Societies, 5th-7th July, Singapore*. [S.l.: s.n.], 2000.

[9] JUANG, J.; CHEN, C.-Y.; YANG, C.-F. *Proceedings of the 2nd International Conference on Intelligent Technologies and Engineering Systems (ICITES2013)*. [S.l.]: Springer Science & Business, 2014.

[10] MIZUNO, H. et al. Application of neural network to technical analysis of stock market prediction. *Studies in Informatic and control*, v. 7, n. 3, p. 111–120, 1998.

[11] KAR, A. Stock prediction using artificial neural networks. *Dept of Computer Science and Engineering, IIT Kanpur*.

[12] CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, Springer, v. 5, n. 4, p. 455–455, 1992.

[13] BENGIO, I. G. Y.; COURVILLE, A. Deep learning. Book in preparation for MIT Press. 2016. Disponível em: <http://www.deeplearningbook.org>.

[14] TAIGMAN, Y. et al. Deepface: Closing the gap to human-level performance in face verification. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. p. 1701–1708.

[15] HAYKIN, S. S. et al. *Neural networks and learning machines*. [S.l.]: Pearson Upper Saddle River, NJ, USA:, 2009.

APÊNDICES

I. TEOREMA DA CONVERGÊNCIA DO PERCEPTRON

A prova para a o algoritmo de convergência do perceptron será apresentada para as seguintes condições:

- Os vetores de entradas são dados por por:

$$\mathbf{x}(n) = \left[\; b, \quad x_1(n), \quad x_2(n), \quad x_3(n), \quad ..., \quad x_m(n) \; \right]^T; \tag{I.1}$$

- O vetores de peso são:

$$\mathbf{w}(n) = \left[\; w_b(n), \quad w_1(n), \quad w_2(n), \quad w_3(n), \quad ..., \quad w_m(n) \; \right]^T; \tag{I.2}$$

- As classes a serem separadas são $\mathscr{C}_1$ e $\mathscr{C}_2$ e serão separadas da seguinte forma:

  $\mathbf{w}^T\mathbf{x} > 0$ para cada vetor de entrada $\mathbf{x}$ pertencente à classe $\mathscr{C}_1$,

  $\mathbf{w}^T\mathbf{x} \leq 0$ para cada vetor de entrada $\mathbf{x}$ pertencente à classe $\mathscr{C}_2$;

- Condição inicial de $\mathbf{w}(0) = 0$.

Suponha que $\mathbf{w}(n)^T\mathbf{x}(n) < 0$ para $n = 1, 2, ...$, e que o vetor $\mathbf{x}(n)$ pertence ao subespaço $\mathscr{H}_1$. O perceptron classifica incorretamente os vetores $\mathbf{x}_1(1)$, $\mathbf{x}_1(2)$, ..., pois viola a primeira suposição sobre a separação das classes. Então, para a constante $\eta = 1$ podemos usar a Equação (2.9) para escrever

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n) \quad \text{para } \mathbf{x}(\text{n}) \text{ pertencente à classe } \mathscr{C}_1 \tag{I.3}$$

Usando a condição inicial $\mathbf{w}(0) = 0$ e resolvendo a equação para $\mathbf{w}(n+1)$, obtém-se o seguinte resultado:

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + ... + \mathbf{x}(n) \tag{I.4}$$

Como é assumido que as classes $\mathscr{C}_1$ e $\mathscr{C}_2$ são linearmente separáveis, existe uma solução para $\mathbf{w}$ para qual $\mathbf{w}^T\mathbf{x} > 0$ para os vetores $\mathbf{x}(1)$, $\mathbf{x}(2)$, ..., pertencentes ao subespaço . Para uma solução $\mathbf{w}_0$ pode-se definir um número positivo α como

$$\alpha = \min_{\mathbf{x}(n) \in \mathscr{H}_1} \mathbf{w}_0^T \mathbf{x}(n) \tag{I.5}$$

Assim, multiplicando a Equação (I.4) pelo vetor $\mathbf{w}_0^T$ encontra-se

$$\mathbf{w}_0^T \mathbf{w}(n+1) = \mathbf{w}_0^T \mathbf{x}(1) + \mathbf{w}_0^T \mathbf{x}(2) + ... + \mathbf{w}_0^T \mathbf{x}(n) \tag{I.6}$$

Pela definição dada na Equação (I.5), tem-se

$$\mathbf{w}_0^T \mathbf{w}(n+1) \geq n\alpha \tag{I.7}$$

Aplicando a desigualdade de Cauchy-Schwarz para $\mathbf{w}_0$ e $\mathbf{w}(n+1)$ chega-se no seguinte resultado

$$\left\|\mathbf{w}_0^T\right\|^2 \|\mathbf{w}(n+1)\|^2 \geq n^2\alpha^2 \tag{I.8}$$

que é equivalente à

$$\|\mathbf{w}(n+1)\|^2 \geq \frac{n^2\alpha^2}{\|\mathbf{w}_0\|^2} \tag{I.9}$$

Esse resultado agora deve ser guardado. Retornando à Equação (I.3), reescreve-se-a da seguinte forma

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k) \quad \text{para } k = 1, ..., n, \text{ e } \mathbf{x}(k) \in \mathscr{H}_1 \tag{I.10}$$

Tirando a norma Euclidiana de ambos os lados da equação, obtém-se

$$\|\mathbf{w}(k+1)\|^2 = \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 + 2\mathbf{w}^T(k)\mathbf{x}(k) \tag{I.11}$$

Como $\mathbf{w}^T(k)\mathbf{x}(k) \leq 0$, encontra-se, a partir da Equação (I.11), que

$$\|\mathbf{w}(k+1)\|^2 - \|\mathbf{w}(k)\|^2 \leq \|\mathbf{x}(k)\|^2 \quad k = 1, ..., n \tag{I.12}$$

Definindo

$$\beta = \max_{\mathbf{x}(k) \in \mathscr{H}_1} \|\mathbf{x}(k)\|^2 \tag{I.13}$$

e usando as condições iniciais e $\mathbf{w}(0) = 0$, encontra-se a seguinte inequação

$$\|\mathbf{w}(n+1)\|^2 \leq \sum_{k=1}^{n} \|\mathbf{x}(k)\|^2 \leq n\beta \tag{I.14}$$

Analisando a Inequação I.14, percebe-se que deve haver um número suficientemente grande n para que ela entreem conflito com a Inequação I.9. Tem que haver, portanto, um valor máximo para n de forma a satisfazer ambas as inequações. Esse valor é a solução da equação

$$\frac{n_{max}^2 \alpha^2}{\|\mathbf{w}_0\|^2} = n_{max}\beta \tag{I.15}$$

Esse resultado prova que se $\eta = 1$, $\mathbf{w}(0) = 0$ e se existir uma solução $\mathbf{w}_0$ que consiga separar $\mathscr{C}_1$ e $\mathscr{C}_2$, existe um limite n_{max} iterações para a adaptação dos pesos do perceptron para o subespaço $\mathscr{H}_1$.

A mesma demonstração utilizada para $\mathscr{H}_1$ também é válida para $\mathscr{H}_2$.

Considerando que $\eta(n)$ o menor inteiro na qual a seguinte desigualdade é verdadeira:

$$\eta(n)\mathbf{x}^T(n)\mathbf{x}(n) > \left|\mathbf{w}^T(n)\mathbf{x}(n)\right| \tag{I.16}$$

Esse procedimento encontra que, se o produto interno $\mathbf{w}^T(n)\mathbf{x}(n)$ possui um sinal incorreto, então $\mathbf{w}^T(n+1)\mathbf{x}(n)$ na iteração $n+1$ terá o sinal correto. Dessa forma, caso $\mathbf{w}^T(n)\mathbf{x}(n)$ tenha um sinal incorreto em n, pode-se modificar a sequencia de treinamento para que $\mathbf{x}(n+1) = \mathbf{x}(n)$. Cada padrão de entrada é apresentado repetidamente até que ele seja classificado corretamente.

Uma condição inicial $\mathbf{w}(0)$ diferente de zero apenas implica variação no número de iterações necessárias para que haja a convergência. A variação depende se as condições iniciais não nulas se aproximam ou se distanciam da solução $\mathbf{w}_0$.

Para o próximo passo será utilizada uma simples função sinal para a ativação do perceptron:

$$sgn(v) = \begin{cases} +1 & \text{se } v \geq 0 \\ -1 & \text{se } v < 0 \end{cases} \tag{I.17}$$

A resposta do perceptron pode ser escrita como

$$y(n) = sgn[\mathbf{w}^T(n)\mathbf{x}(n)] \tag{I.18}$$

Para calcular o erro é necessário um vetor que represente o valor desejado

$$d(n) = \begin{cases} +1 & \text{se } x(n) \text{ pertence a classe } \mathscr{C}_1 \\ -1 & \text{se } x(n) \text{ pertence a classe } \mathscr{C}_2 \end{cases} \tag{I.19}$$

O erro do sistema é portanto

$$erro = d(n) - y(n) \tag{I.20}$$

Por fim, a adaptação dos pesos $\mathbf{w}(n)$ pode ser concatenada na forma da seguinte regra de aprendizagem para correção de erro[15]

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n) \tag{I.21}$$

II. BACK-PROPAGATION

Considere a Figura II.1, que representa um neurônio j sendo alimentado por um conjunto de sinais oriundos da uma camada de neurônios à sua esquerda.

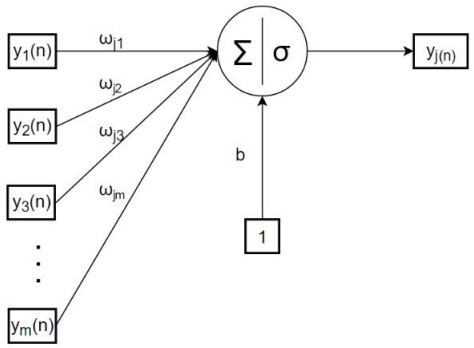

Figura II.1: Gráfico que representa a propagação do sinal em um neurônio j

O campo local produzido na entrada da função de ativação do neurônio j é

$$s_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{II.1}$$

aonde m é o número total de entradas (exceto pelo bias) aplicado ao neurônio m.

O algoritmo de *back-propagation* aplica uma correlação entre $\Delta w_{ji}(n)$ e o ajuste $w_{ji}(n)$, proporcional a derivada parcial $\partial \mathscr{E} / \partial w_{ji}(n)$. De acordo com a regra da cadeia, pode-se expressar o gradiente como

$$\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathscr{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial s_j(n)} \frac{\partial s_j(n)}{\partial w_{ji}(n)} \tag{II.2}$$

A derivada parcial $\partial \mathscr{E} / \partial w_{ji}(n)$ representa a sensibilidade do sistema e determina a direção da busca no espaço de pesos para o peso sináptico w_{ji}.

Somando todos os erros na camada de saída da rede, tem-se o erro total instantâneo do sistema.

$$\mathscr{E}(n) = \sum_{j \in C} \mathscr{E}_j(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n) \tag{II.3}$$

Em que C inclui todos os neurônios da camada de saída.

Diferenciando os dois lados da equação II.3 em respeito a $e_j(n)$ encontra-se

$$\frac{\partial \mathscr{E}(n)}{\partial e_j(n)} = e_j(n) \tag{II.4}$$

Para $y_j(n) = o_j(n)$, diferenciando os dois lados da equação 2.14 encontra-se

$$\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} = -1 \tag{II.5}$$

Em seguida, diferenciando a equação 2.11 em respeito à $s_j(n)$, tem-se

$$\frac{\partial y_j(n)}{\partial s_j(n)} = \sigma_j'(s_j(n)) \tag{II.6}$$

Por fim, diferenciando a equação II.1 em relação à $w_{ji}(n)$ chega-se em

$$\frac{\partial s_j(n)}{\partial w_{ji}(n)} = y_i(n) \tag{II.7}$$

Juntando as equações II.7 e II.4 na equação II.2, encontra-se a expressão

$$\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} = -e_j(n)\sigma_j'(s_j(n))y_i(n) \tag{II.8}$$

O ajuste $\Delta w_{ji}(n)$ aplicado em $w_{ji}(n)$ é definida por

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} \tag{II.9}$$

com η sendo o parâmetro de aprendizagem no algoritmo de *back-propagation*. O sinal negativo na equação II.9 representa a queda no gradiente, significando uma mudança de pesos em direção ao menor valor de $\mathscr{E}(n)$.

Substituindo a equação II.8 na equação II.9:

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{II.10}$$

aonde o gradiente local $\delta_j(n)$ é definida por

$$\begin{aligned} \delta_j(n) &= \frac{\partial \mathscr{E}(n)}{\partial s_j(n)} \\ &= \frac{\partial \mathscr{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial s_j(n)} \\ &= e_j(n)\sigma_j'(s_j(n)) \end{aligned} \tag{II.11}$$

Agora encontrar o gradiente local da última camada é trivial, sabendo o erro $e_j(n)$ basta aplicar a equação II.11.

Para determinar o gradiente de camadas escondidas, a tarefa é um pouco mais complexa. Um neurônio não pertencente à camada de saída não tem uma saída esperada específica. Portanto, o sinal de erro de um neurônio escondido é determinado propagando-se para trás o sinal de erro de todos os neurônios dos quais ele está conectado. Esse processo é recursivo, logo, um neurônio da

camada $[l]$ vai ter como em seu sinal de erro, influência de todos os erros da camada $[l+1]$, o sinal de erro de cada neurônio dessa camada terá, por sua vez, influência por parte de cada neurônio da camada $[l+2]$ e assim sucessivamente.

Para para definir como essa propagação ocorre, primeiro será utilizada a seguinte redefinição do gradiente $\delta_j(n)$ para um neurônio escondido j

$$\begin{aligned}\delta_j(n) &= -\frac{\partial \mathscr{E}(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial s_j(n)} \\ &= -\frac{\partial \mathscr{E}(n)}{\partial y_j(n)}\sigma_j'(s_j(n))\end{aligned} \tag{II.12}$$

Utilizando a equação II.3 para um neurônio escondido j seguido de um outro neurônio, da camada de saída, k, de tal forma que j está presente na camada l e k está presente na camada L, tem-se

$$\mathscr{E}(n) = \frac{1}{2}\sum_{j\in C} e_j^2(n) \tag{II.13}$$

Diferenciando a equação II.13 em relação ao sinal $y_j(n)$, e aplicando a regra da cadeia, resulta-se em

$$\begin{aligned}\frac{\partial \mathscr{E}}{\partial y_i(n)} &= \sum_k e_k \frac{\partial e_k(n)}{\partial y_i(n)} \\ &= \sum_k e_k \frac{\partial e_k(n)}{\partial s_i(n)}\frac{\partial s_k(n)}{\partial y_i(n)}\end{aligned} \tag{II.14}$$

Para o neurônio de saída k

$$\begin{aligned}e_k(n) &= d_k - y_k(n) \\ &= d_k(n) - \sigma_k(s_k(n))\end{aligned} \tag{II.15}$$

Assim,

$$\frac{\partial e_k(n)}{\partial s_k(n)} = -\sigma_k'(s_k(n)) \tag{II.16}$$

Também é notável que, para esse mesmo neurônio k, o campo local é

$$s_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n) \tag{II.17}$$

em que m é o total de entradas, excluindo o bias, aplicado ao neurônio k. Diferenciando a equação II.17 em respeito a $y_j(n)$, tem-se

$$\frac{\partial s_k(n)}{\partial y_k(n)} = w_{kj}(n) \tag{II.18}$$

Substituindo as equações II.16 e II.18 na equação II.14 tem-se

$$\frac{\partial \mathscr{E}}{\partial y_i(n)} = -\sum_k e_k(n)\sigma_k'(s_k(n))w_{kj}(n) \tag{II.19}$$

que pode ser simplificada como

$$\frac{\partial \mathscr{E}}{\partial y_i(n)} = -\sum_k \delta_k(n)w_{kj}(n) \tag{II.20}$$

A Equação (II.20) é a definição do gradiente local $\delta_k(n)$, similar à Equação (II.11), mas com o índice j substituído por k. Finalmente, utilizando a Equação (II.20) na Equação (II.12) encontra-se a fórmula do *back-propagation.*

$$\delta_j(n) = \sigma_j'(s_j(n))\sum_k \delta_k(n)w_{kj}(n), \quad \textit{para um neurônio } j \textit{ escondido} \tag{II.21}$$

A generalização do algoritmo para a correção $\Delta w_{ji}(n)$ aplicada ao peso sináptico conectando o neurônio i ao neurônio j é

$$\begin{pmatrix} \text{correção} \\ \text{de pesos} \\ \Delta w_{ji(n)} \end{pmatrix} = \begin{pmatrix} \text{parâmetro de} \\ \text{aprendizagem} \\ \eta \end{pmatrix} \times \begin{pmatrix} \text{gradiente} \\ \text{local} \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} \text{sinal de entrada} \\ \text{do neurônio } j, \\ y_i(n) \end{pmatrix} \tag{II.22}$$

na qual $\delta_j(n)$ depende se o neurônio j é um neurônio de saída ou é um neurônio escondido. Portanto, a forma geral de $\delta_j(n)$ é[15]

$$\delta_j(n) = \begin{cases} e_j(n)\sigma_j'(s_j(n)), & \text{se o neurônio } j \text{ for um neurônio da camada de saída,} \\ \sigma_j'(s_j(n))\sum_k \delta_k(n)w_{kj}(n), & \text{se o neurônio } j \text{ for um neurônio de camada escondida.} \end{cases} \tag{II.23}$$

III. CORRELAÇÃO CRUZADA ENTRE COTAÇÃO E POPULARIDADE NO GOOGLE TRENDS

Esse apêndice contém, listadas, os gráficos das correlações cruzadas entre as cotações do ativo PETR4 da Petrobras e o número de pesquisas dos diferentes temas utilizados no trabalho.

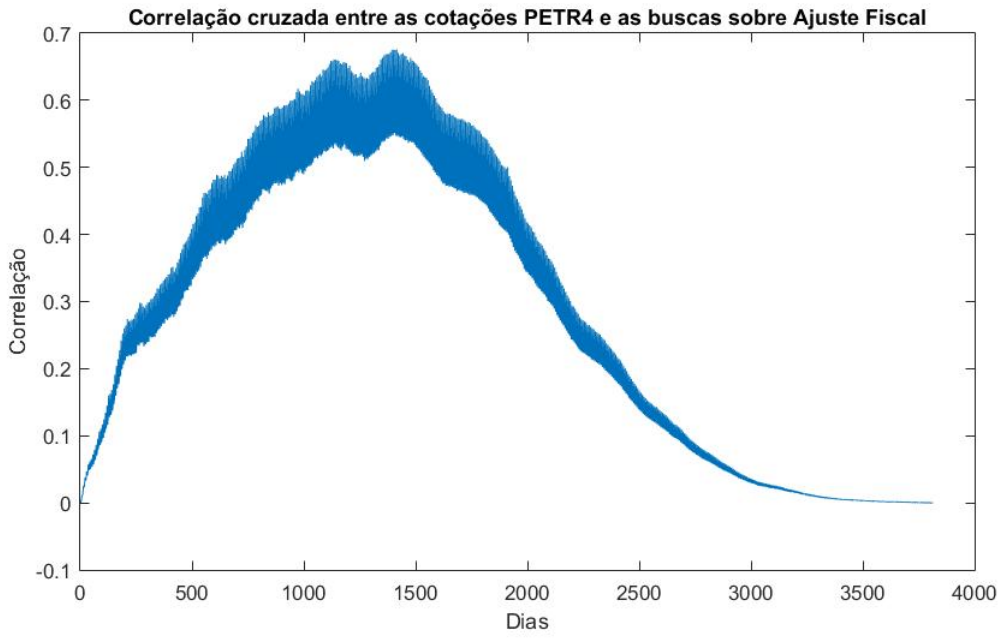

Figura III.1: Correlação cruzada entre contações e "Ajuste Fiscal"

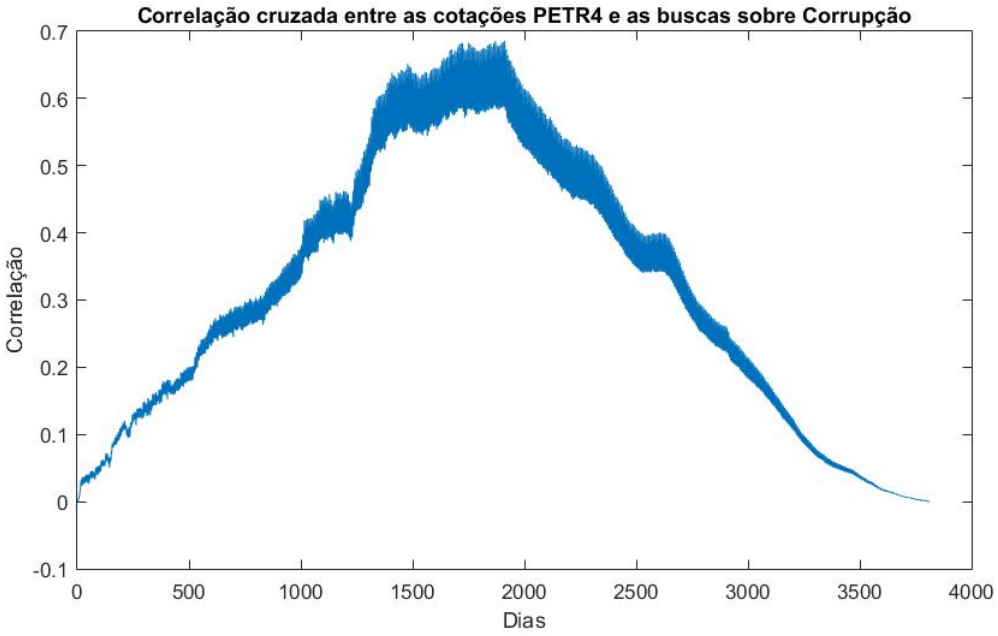

Figura III.2: Correlação cruzada entre contações e "Corrupção"

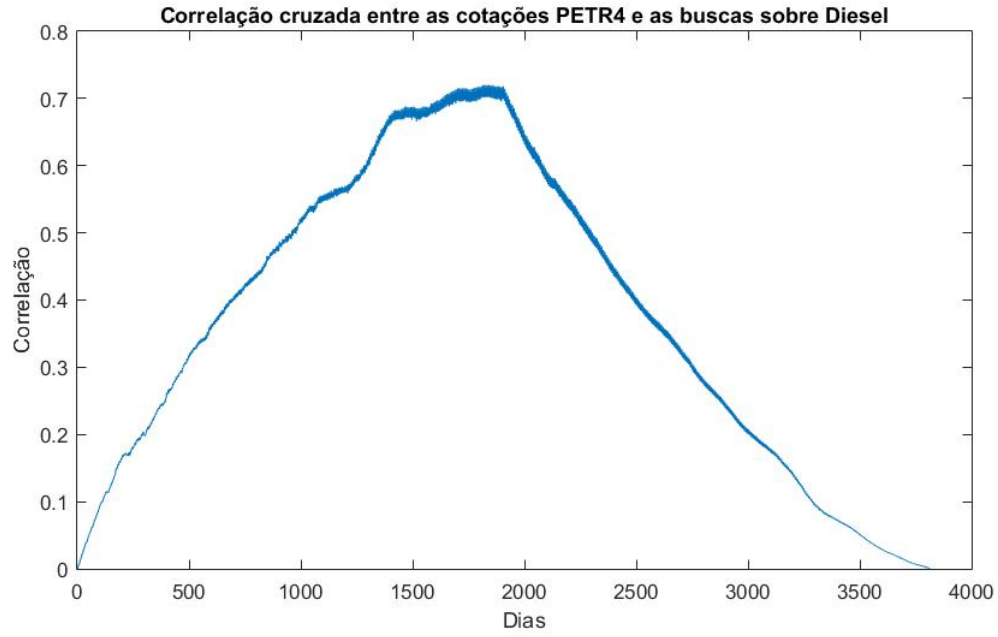

Figura III.3: Correlação cruzada entre contações e "Diesel"

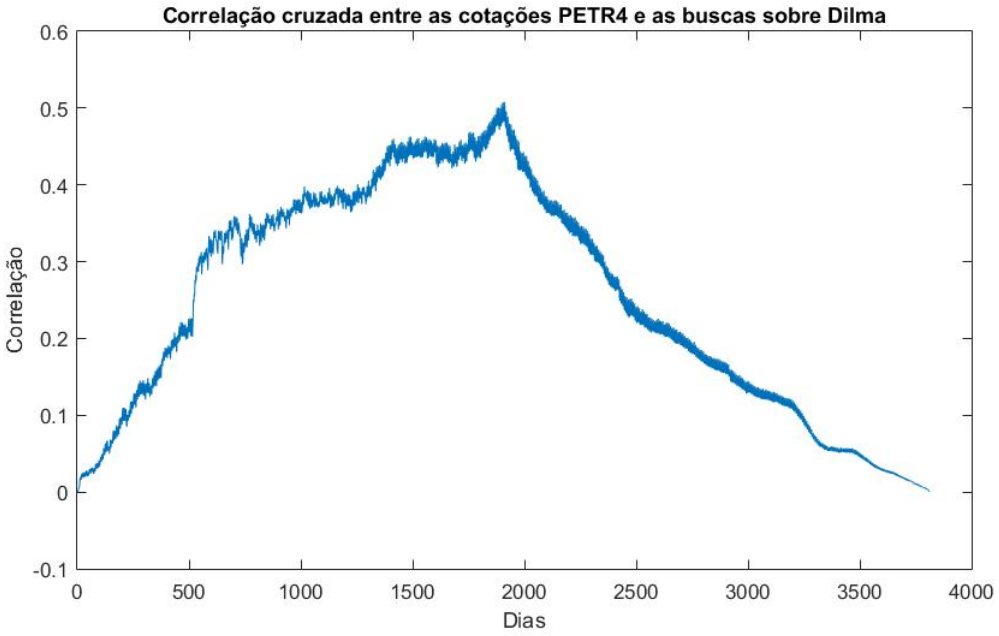

Figura III.4: Correlação cruzada entre contações e "Dilma"

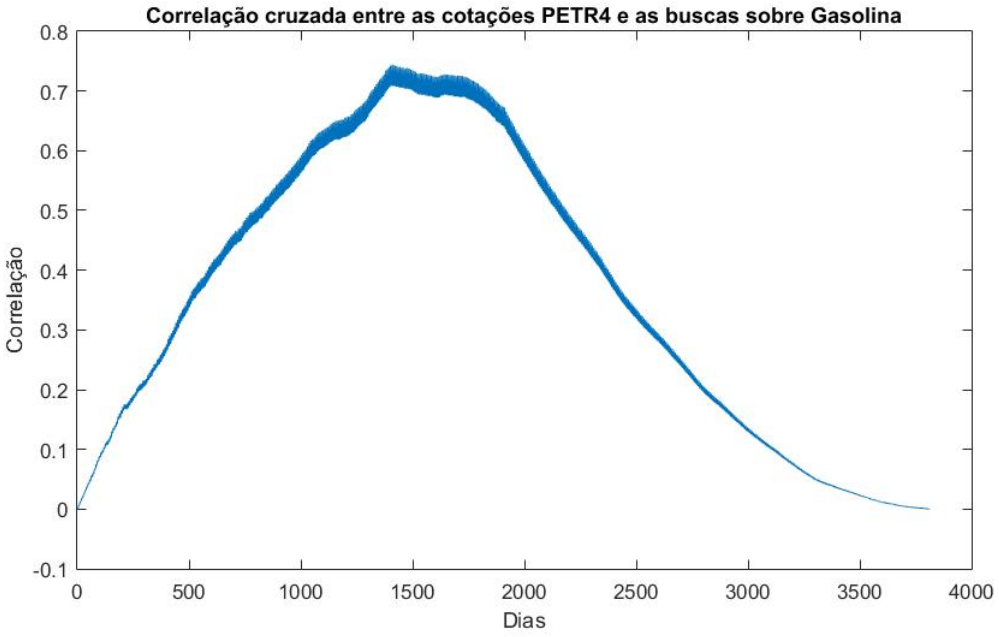

Figura III.5: Correlação cruzada entre contações e "Gasolina"

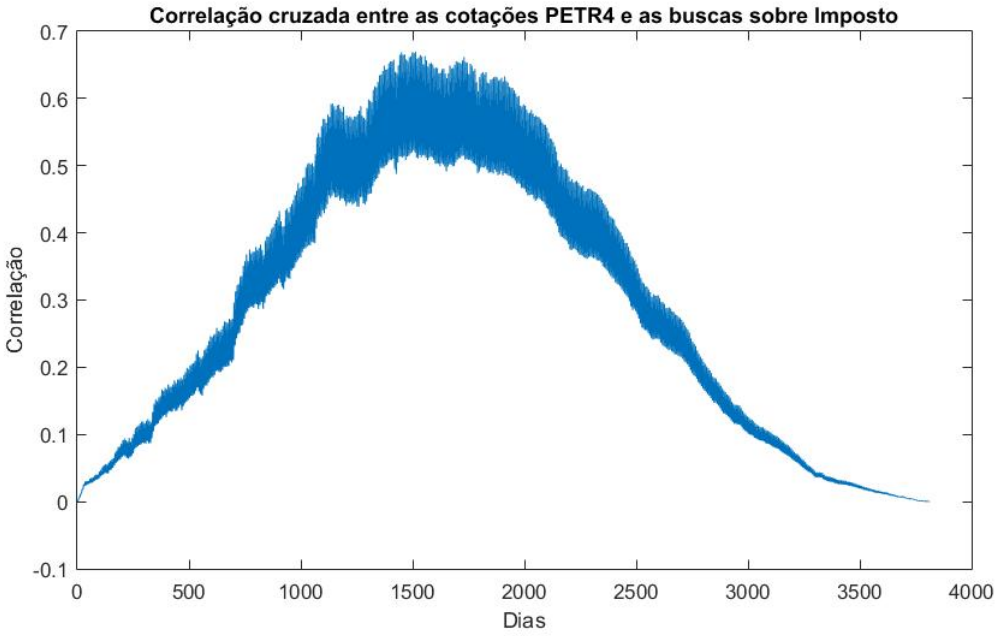

Figura III.6: Correlação cruzada entre contações e "Imposto"

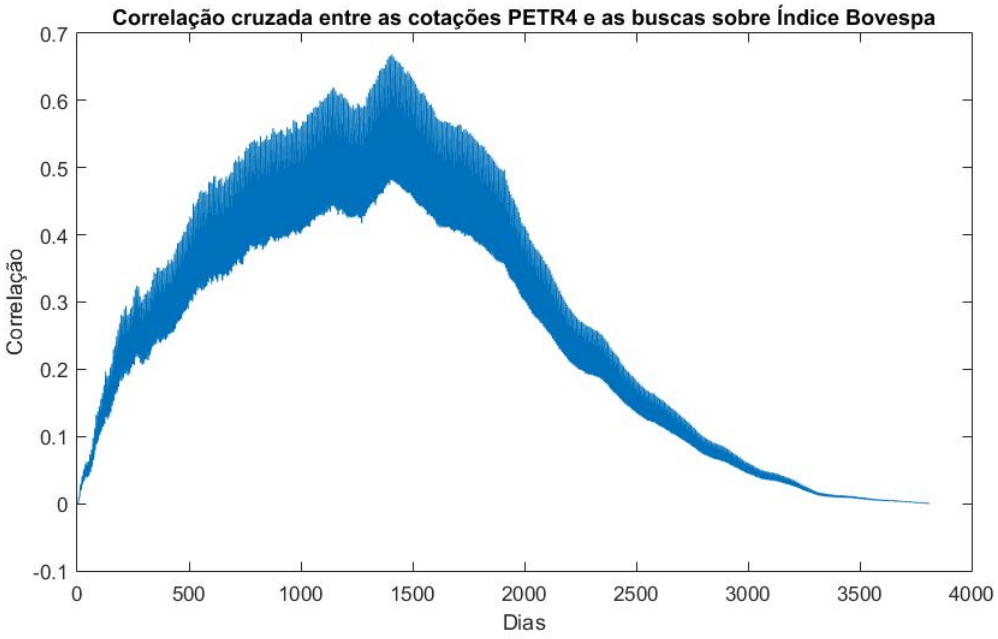

Figura III.7: Correlação cruzada entre contações e "Índice Bovespa"

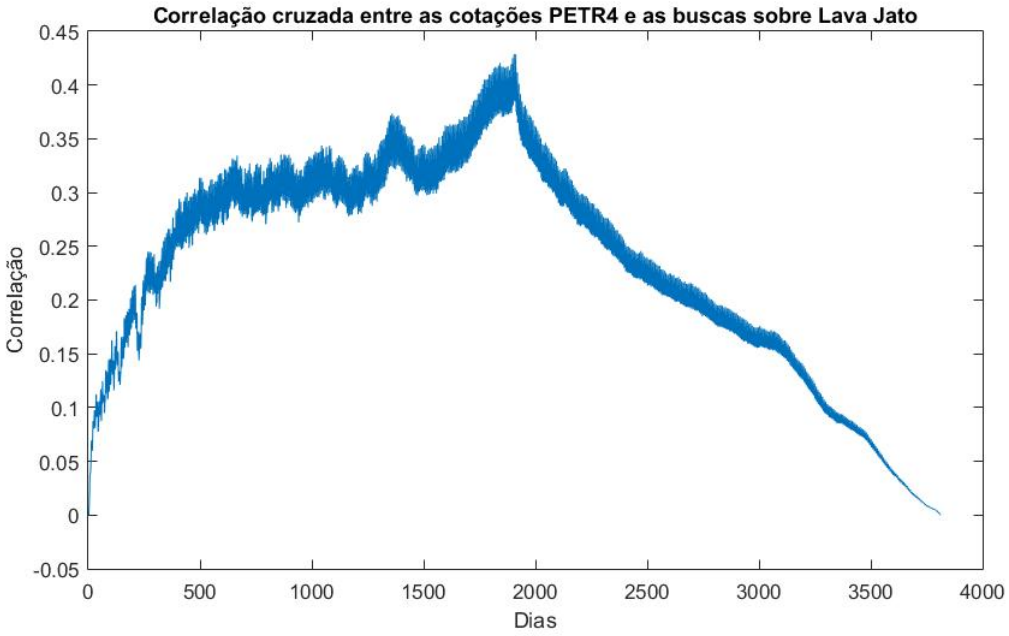

Figura III.8: Correlação cruzada entre contações e "Lava Jato"

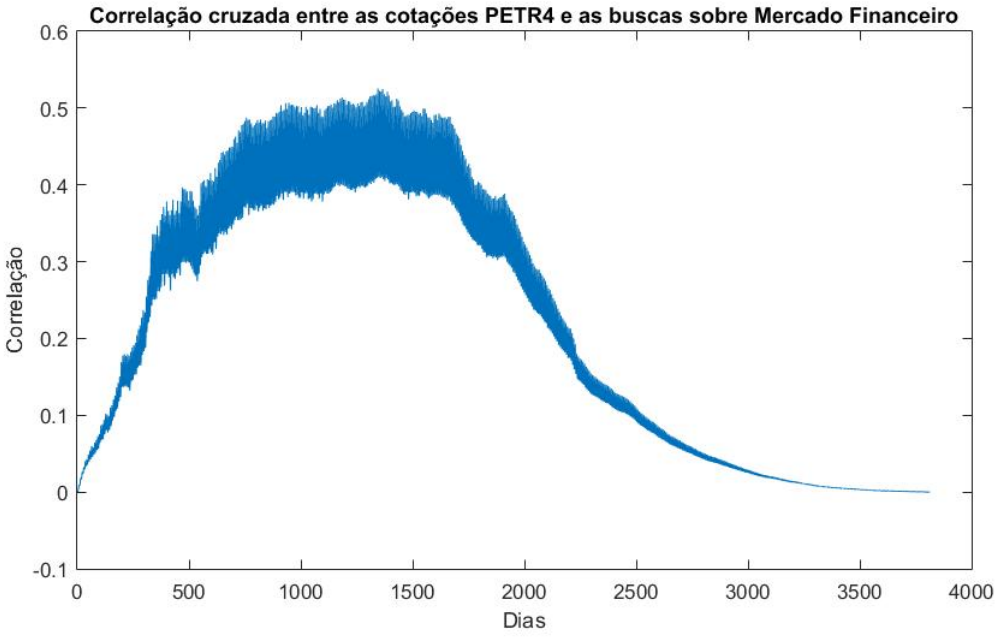

Figura III.9: Correlação cruzada entre contações e "Mercado Financeiro"

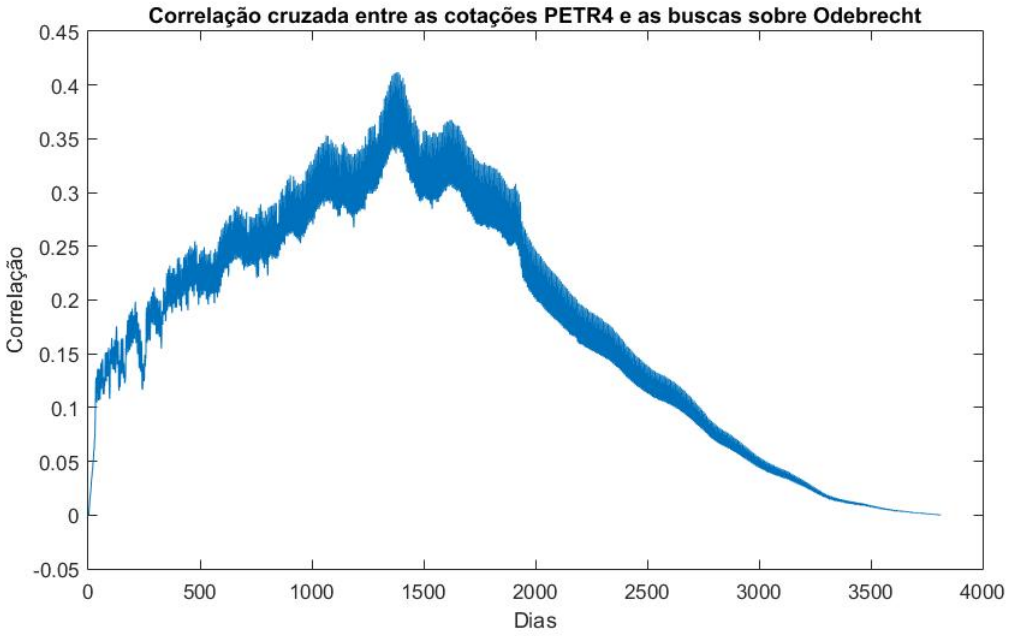

Figura III.10: Correlação cruzada entre contações e "Odebrecht"

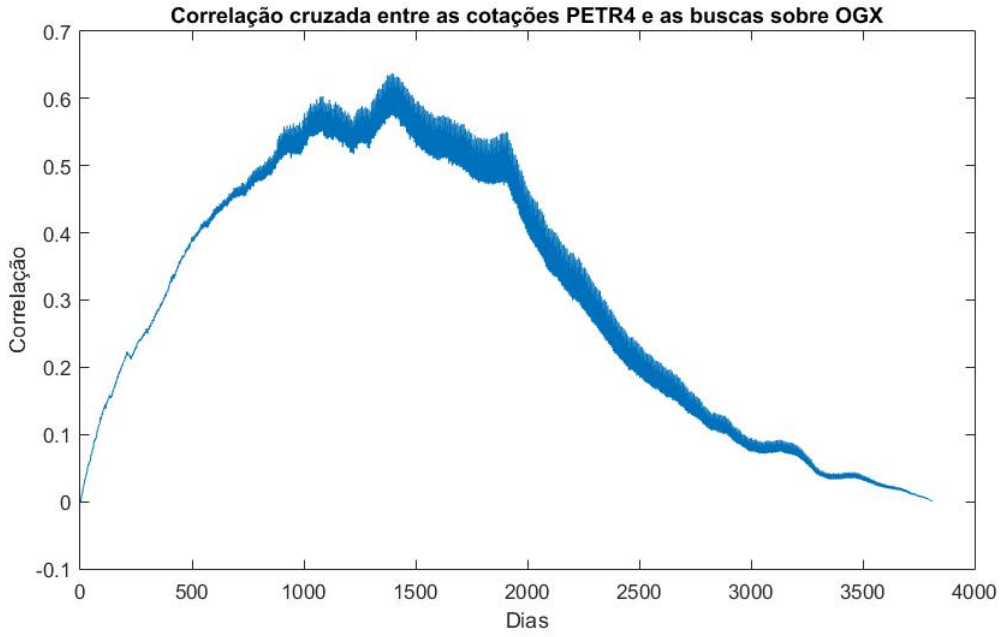

Figura III.11: Correlação cruzada entre contações e "OGX"

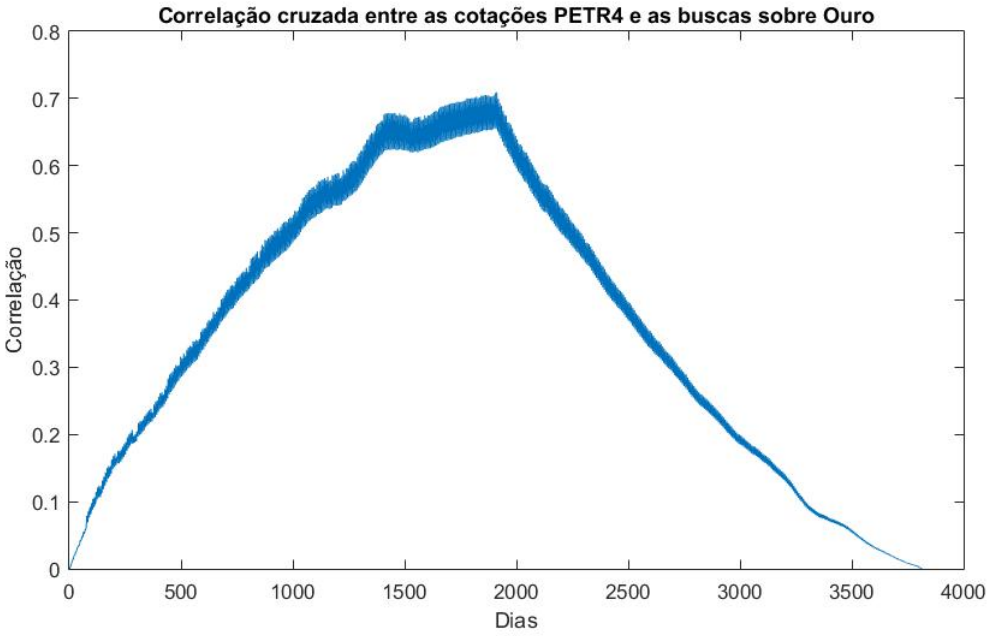

Figura III.12: Correlação cruzada entre contações e "Ouro"

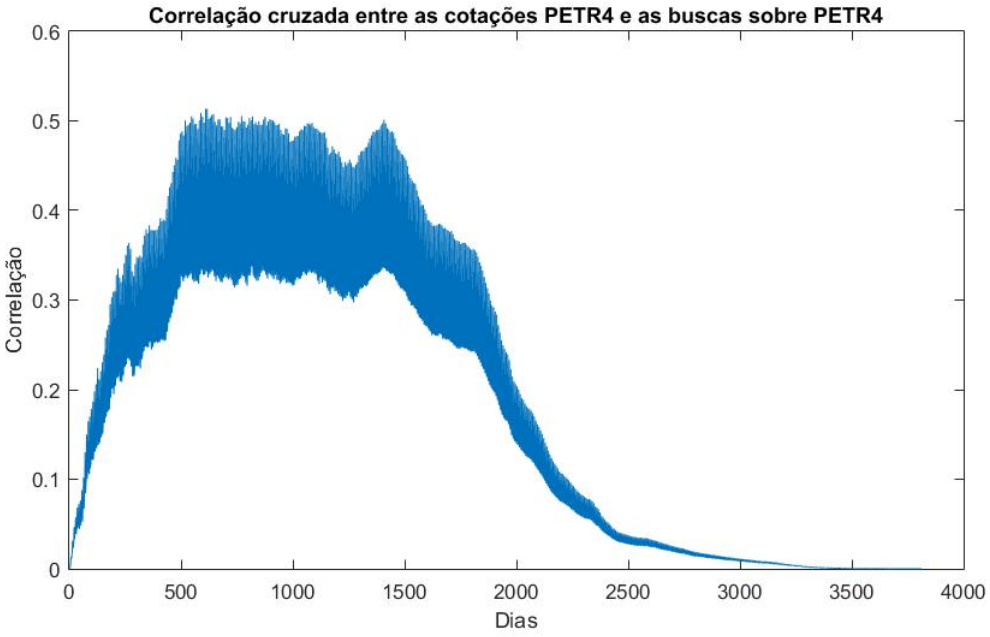

Figura III.13: Correlação cruzada entre contações e "PETR4"

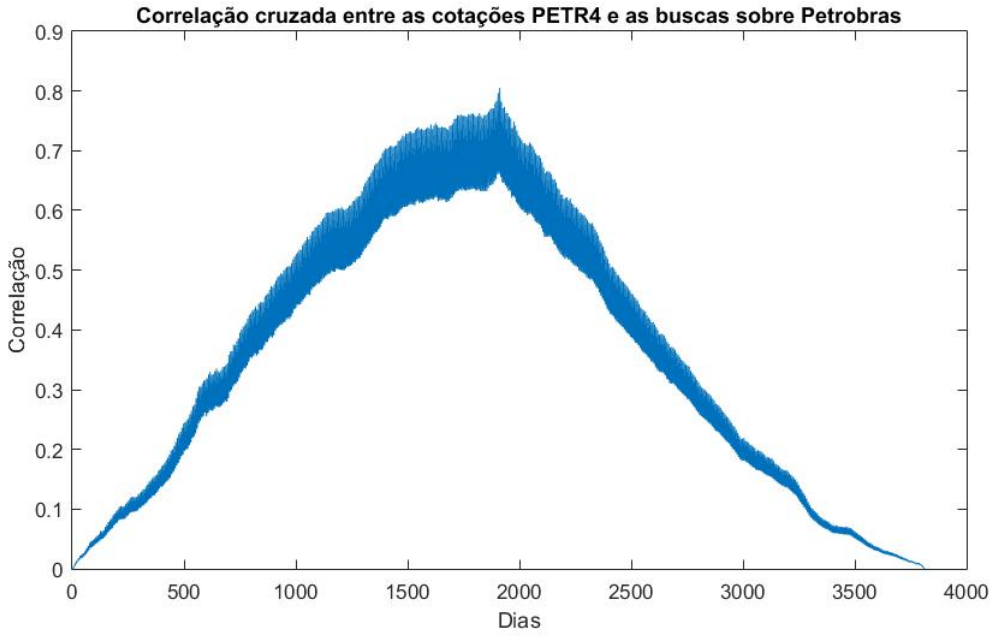

Figura III.14: Correlação cruzada entre contações e "Petrobras"

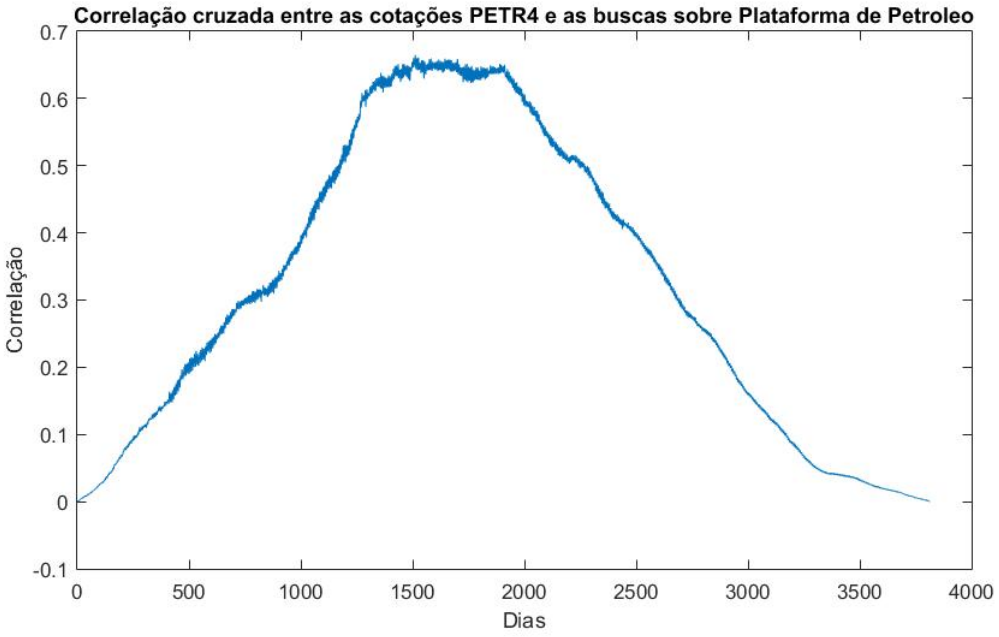

Figura III.15: Correlação cruzada entre contações e "Plataforma de Petróleo"

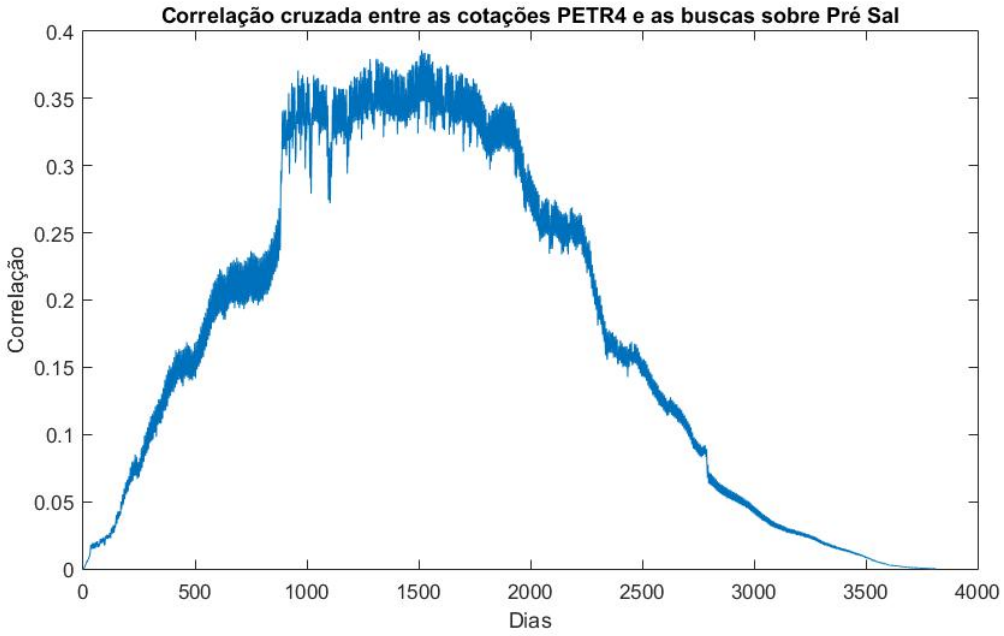

Figura III.16: Correlação cruzada entre contações e "Pré Sal"

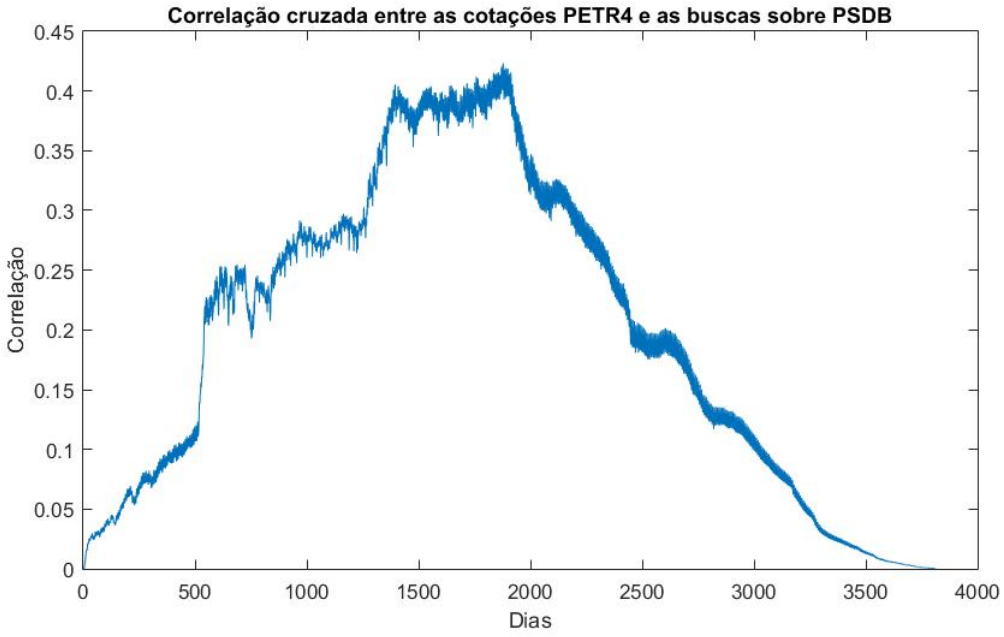

Figura III.17: Correlação cruzada entre contações e "PSDB"

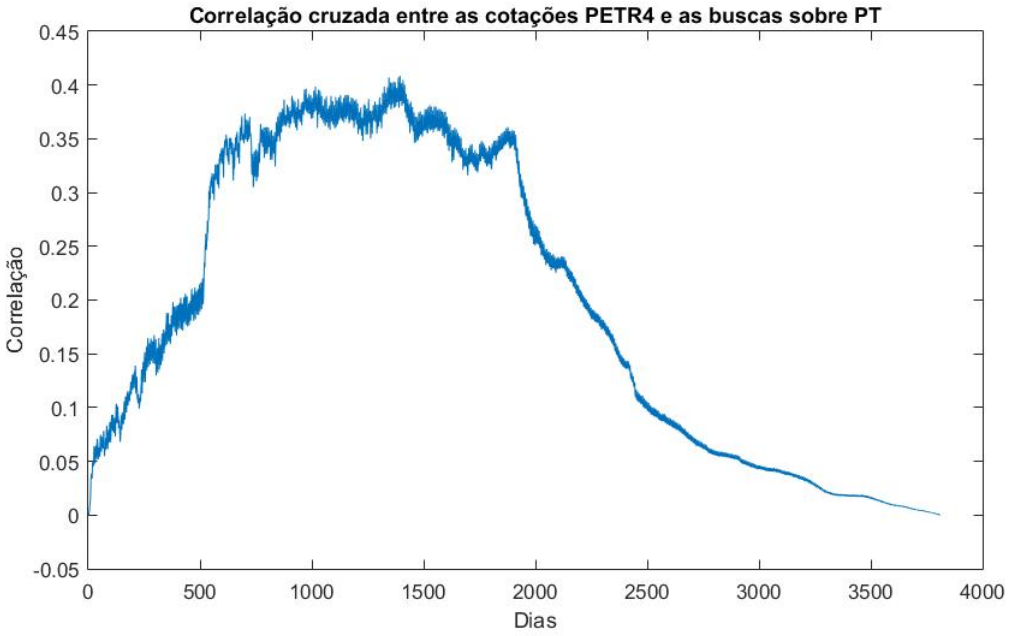

Figura III.18: Correlação cruzada entre contações e "PT"

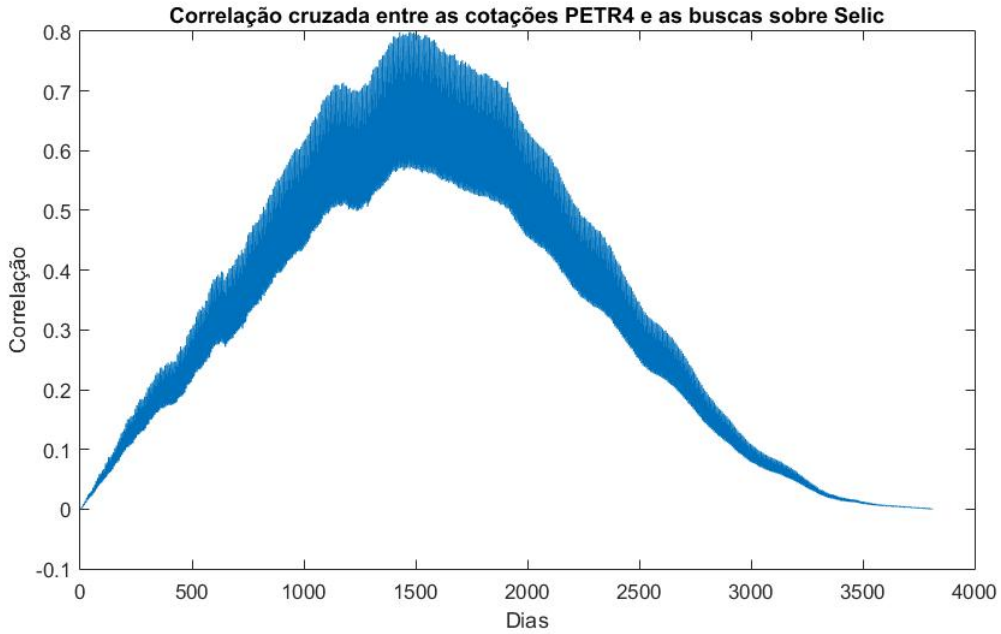

Figura III.19: Correlação cruzada entre contações e "Selic"

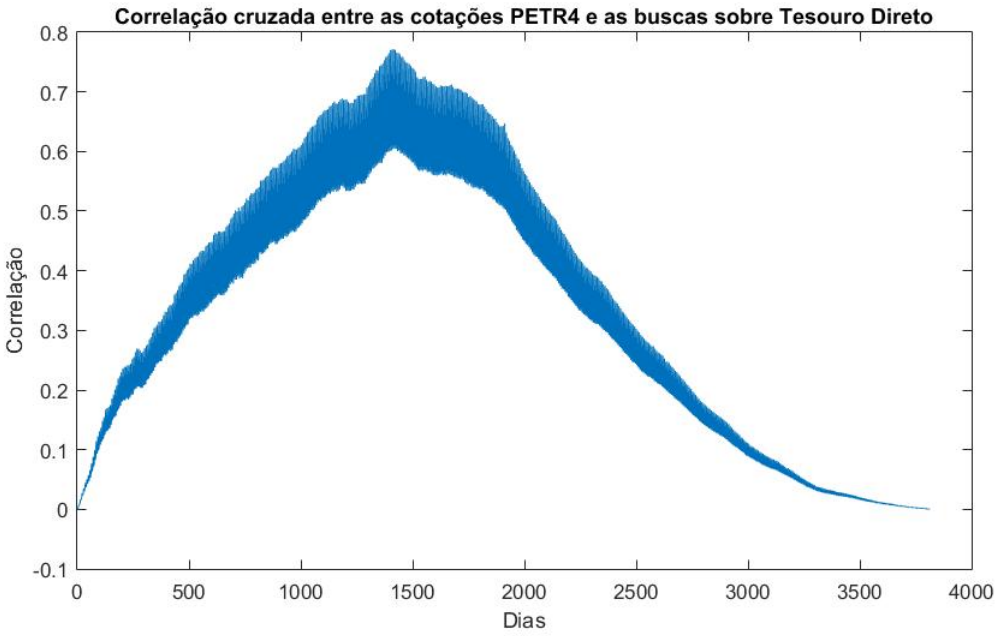

Figura III.20: Correlação cruzada entre contações e "Terouro Direto"