MAE 5905: Introdução à Ciência de Dados

Pedro A. Morettin

Instituto de Matemática e Estatística
Universidade de São Paulo
pam@ime.usp.br
http://www.ime.usp.br/∼ pam

Aula 13

4 de maio de 2026

Sumário

1. Deep Learning
2. Redes Neurais Recorrentes
3. Redes Neurais Long-Short-Term Memory - LSTM

Deep learning

- Como vimos anteriormente, ML (aprendizado DE máquina) envolve procedimentos segundo os quais um sistema computacional adquire a habilidade de aprender extraindo padrões de dados. Ou seja, tanto na programação clássica, como em ML, nada é conseguido sem dados, que podem tomar a forma de números, vetores, curvas, imagens, palavras etc.
- Vimos que o desempenho de um algoritmo de ML depende da representação desses dados, por meio de funções pré-determinadas, como a logística, tangente hiperbólica etc. A escolha dessa representação é crucial.
- No caso de termos várias camadas intermediárias obtém-se o que é chamado aprendizado profundo (deep learning - DL). Nesse caso, o sistema computacional (SC) aprende a partir dos dados (e exemplos) em termos de uma hierarquia de conceitos, cada um definido por meio de sua relação com conceitos mais simples, ou ainda, o SC aprende conceitos complicados a partir de conceitos simples.

Deep learning

- Um exemplo de modelo DL é o multilayer perceptron, ou seja, uma aplicação que leva os dados de entrada a valores de saída, por meio de composição de funções mais simples.
- Segundo Goodfellow et al. (2016), modelos DL remontam à década de 1940, com os nomes de Cibernética (1940-1960), coneccionismo acoplado a redes neurais (1980-1990) e ressurge como DL em 2006.
- Por sua vez, os tamanhos de conjuntos de dados expandiram-se de small data, da ordem de centenas ou milhares na década de 1900–1980, a dezenas ou centenas de milhares de dados após 1990 (big data). Um exemplo bastante estudado na área é o MNIST (Modified National Institute of Standards and Technology), que consiste de fotos de dígitos escritos a mão.

Deep learning

- A partir de 2000, conjuntos de dados ainda maiores surgiram, contendo dezenas de milhões de dados, notadamente dados obtidos na Internet. Para analisar esses megadados foi necessário o desenvolvimento de CPU's mais rápidas e as chamadas GPU's (Graphics Processing Units). Essas são usadas em celulares, computadores pessoais, estações de trabalho e consoles de jogos e são muito eficientes em computação gráfica e processamento de imagens. Sua estrutura altamente paralela as torna eficientes para algoritmos que processam grandes blocos de dados em paralelo.
- A complexidade de um algoritmo é proporcional ao número de observações, número de preditores, número de camadas e número de épocas de treinamento. Para detalhes sobre esses tópicos, veja Hastie et al. (2017) e Chollet (2018).

Deep learning

Como dissemos acima, DL tornou-se proeminente nas décadas de 2006–2010. Conforme Chollet (2018), as seguintes aplicações tornaram-se possíveis:

- classificação de imagens ao nível quase humano (NQH);
- reconhecimento de falas ao NQH;
- transcrições de escritas a mão ao NQH;
- aperfeiçoamento de técnicas de ML;
- aperfeiçoamento de conversões textos-para-falas;
- carros autônomos ao NQH;
- aperfeiçoamento de buscas na Internet;
- assistentes digitais, como o Google Now e Amazon Alexa;
- habilidade para responder questões sobre linguagens naturais.

Deep learning

- Todavia, muitas aplicações levarão muito tempo para serem factíveis e questões relativas ao desenvolvimento, por SC, da inteligência ao nível humano, não devem ser seriamente consideradas no presente estágio de desenvolvimento. Em particular, muitas afirmações feitas nas décadas de 1960–1970 e 1980–1985 sobre expert systems e AI em geral, que não se concretizaram, levaram a um decréscimo de investimento em pesquisa na área.
- Em termos de software, há diversas bibliotecas para ML e DL, como Theano, Torch, MXNet e TensorFlow.

Deep learning

- Para analisar redes neurais com várias camadas é possível usar o pacote keras do R, que por sua vez usa o pacote Tensorflow com capacidades CPU e GPU. Modelos *deep learning* que podem ser analisados incluem redes neurais recorrentes (*recurrent neural networks* - RNN), redes Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN)etc.
- As redes LSTM são apropriadas para captar dependências de longo prazo e implementar previsão de séries temporais.
- As redes `CNN` representam o estado da arte atualmente, usadas para classificação de imagens, linguagens naturais e outras aplicações.
- Para uma excelente revisão sobre aprendizado profundo veja o artigo de LeCun et al. (2015) e as referências nele contidas, sobre os principais desenvolvimentos na área até 2015. A seguir daremos breves introduções sobre RNN e CNN, com alguns exemplos.

Redes neurais recorrentes - RNN

- O termo RNN é usado indiscriminadamente para se referir a classes amplas de redes com uma estrutura geral similar. Essas redes exibem um comportamento temporal dinâmico.
- RNN foram baseadas nos trabalhos de David Rumelhart em 1986 e John Hopfield em 1982. LSTM foram inventadas por Hochreiter e Schmidhuber em 1997 e estabeleceram recordes de acurácia em diversas aplicações.
- RNN tem sido um foco importante de pesquisa desde a década de 1990. Elas são planejadas para aprender padrões sequenciais ou variando no tempo.
- RNN têm sido aplicada a uma variedade grande de problemas. No final da década de 1980 foram introduzidas RNN parciais por vários pesquisadores, como Rumelhart, Hinton e Williams (veja Rumelhart, 1986) para aprender sequências de caracteres. Outras aplicações incluem problemas em sistemas dinâmicos com sequências temporais de eventos, estimaçãoda potência de turbinas, previsões financeiras, sinteses musicais, previsões de de cargas elétricas, previsão de vazões de rios etc.

Redes neurais recorrentes - RNN

- A arquitetura de RNN pode ser:

  (a) completamente interconectada: todos os estados são alcançados por qualquer estado, inclusive cada estado de si próprio;

  (b) parcialmente conectada: nem todos os estados são alcançados da partir de um estado qualquer. Essas redes são usadas para aprender sequência de caracteres.

- Pesquisadores desenvolveram uma variedade de esquemas pelos quais métodos gradiente, e em especial, aprendizado por backpropagation, podem ser estendidos a RNN. Werbos introduziu um procedimento que aproxima a evolução de uma RNN como uma sequência de redes estáticas usando métodos gradiente (*backpropagation through time*). Outras abordagens podem ser encontradas em Lapedes e Farber(1986), Pineda (1987), Almeida (1987), Sato (1990). As várias tentativas para estender backpropagation para RNN estão sumarizadas em Pearlmutter (1995).

Redes neurais recorrentes - RNN

- Do ponto de vista de séries temporais, RNN podem ser vistas como modelos não lineares de espaços de estados. Veja Morettin e Toloi (2020) para detalhes sobre esses modelos.
- Uma RNN processa um elemento por vez de uma sequência de entrada X_t, mantendo sua unidades ocultas num **vetor de estados** S_t, que contém informação sobre a história passada da sequência (LeCun et al. 2015). Denotando por Y_t a sequência processada no estado t, uma rede neural recorrente processa sequências de entradas iterativamente,

$$Y_t = f(Y_{t-1}, X_t),$$

e as previsões são feitas segundo

$$\widehat{Y}_{t+h|t} = g(h_t),$$

com f e g denotando funções a serem definidas.
- Todas as redes neurais recorrentes têm a forma de uma rede neural que se repete como na Figura 1: A, B e W são parâmetros (matrizes) invariantes no tempo. O algoritmo BP pode ser aplicado à rede desdobrada.

Redes neurais recorrentes - RNN

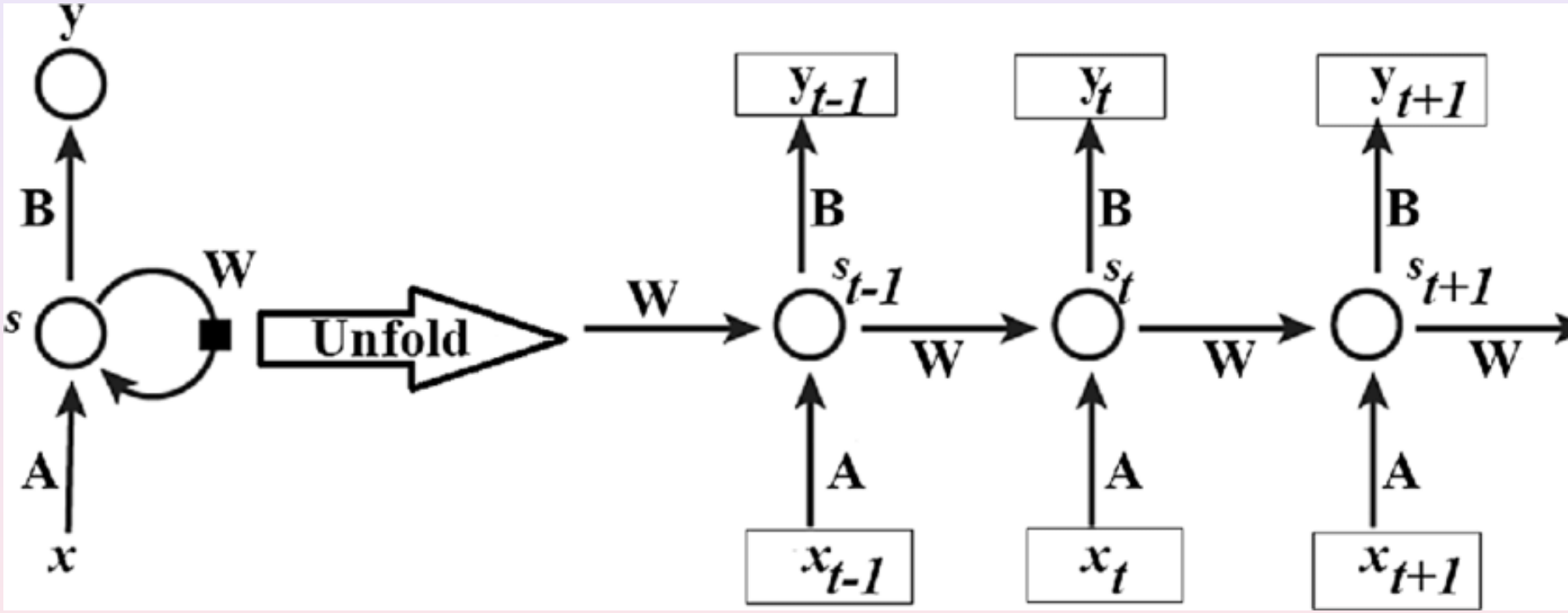


Figura 1: RNN e seu desdobramento no tempo.

Redes neurais recorrentes - RNN

- Redes neurais recorrentes são difíceis de implementar pois sofrem do chamado **problema do gradiente evanescente** (*vanishing gradient problem*).
- Uma solução foi proposta por Hochreiter e Schmidhuber (1997) por meio de uma variante das redes neurais recorrentes, chamada **rede com memórias de curto e longo prazos** (*Long-Short-Term Memory - LSTM*), capazes de "aprender" dependências de longo prazo.
- No caso de séries temporais, há modelos para descrever processos com memória curta (*short memory*), como os modelos *ARIMA* (autorregressivos, integrados e de média móveis), e modelos para descrever memória longa (*long memory*), como os modelos *ARFIMA* (autorregressivos, integrados fracionários e de médias móveis). Veja, por exemplo, Morettin e Toloi (2018).
- Uma rede *LSTM* modela simultaneamente as memórias de curto e longo prazos. Discutimos brevemente essas redes na seção seguinte.

LSTM

- Cerca de 2007, LSTMs revolucionaram o reconhecimento de fala, superando modelos tradicionais em aplicações nessa área. Em 2009, uma rede **Connectionist Temporal Classification (CTC)-trained LSTM** foi a primeira RNN a vencer competições em reconhecimento de padrões.
- LSTMs também melhoraram o reconhecimento de grandes vocabulários de falas e sínteses de textos para falas e foi usada no Google Android. Em 2015, o reconhecimento de falas do Google obteve um salto de desempenho da ordem de 49% por meio de uma rede **CTC-trained LSTM**.
- Redes LSTM quebraram recordes para melhorar traduções de textos, modelagem de linguagens e processamento de múltiplas linguas. LSTM combinada com **convolutional neural networks (CNNs)** melhoraram a captação automática de imagens.

LSTM

- Uma rede LSTM consiste de blocos de memória, chamadas **células** (*cells*), conectadas por meio de camadas (*layers*).
- A informação nas células está contida no estado C_t e no estado escondido h_t e é regulada por mecanismos chamados **portas** (*gates*), por meio de funções de ativação *sigmoid* e *tanh*.
- A função sigmoide tem como saídas números entre 0 e 1, com 0 indicando *nada passa* e 1 indicando *tudo passa*. A rede LSTM tem, portanto, a habilidade de adicionar ou desprezar informação do estado da célula (condicionalmente).
- Em geral, as portas têm como entradas os estados escondidos do instante anterior, h_{t-1} e da entrada atual x_t e as multiplica por pesos matriciais, $\mathbf{W}$, e um viés (*bias*) b é adicionado ao produto.

LSTM

Há três portas principais:

(i) **Forget gate**: esta porta determina qual informação será desprezada do estado da célula. A saída será 0, significando **apagar** e 1, implicando **lembrar todos**; formalmente,

$$f_t = \sigma\left(W_f[h_{t-1}, x_t] + b_f\right).$$

(ii) **Input gate**: nesse passo, a função de ativação *tanh* criará um vetor de candidatos potenciais como segue:

$$\hat{C}_t = \tanh\left(W_c[h_{t-1}, x_t] + b_c\right).$$

A camada sigmoide cria um filtro atual como segue:

$$U_t = \sigma\left(W_u[h_{t-1}, x_t] + b_u\right).$$

A seguir o estado anterior C_{t-1} é atualizado como

$$C_t = f_t C_{t-1} + U_t \hat{C}_t.$$

LSTM

(iii) **Output gate**: nesse passo, a camada sigmoide filtra o estado que vai para a saída:

$$O_t = \sigma\left(W_0[h_{t-1}, x_t] + b_0\right).$$

(iv) O estado C_t é então passado por meio da função *tanh* para escalonar os valores para o intervalo $[-1, 1]$. Finalmente, o estado escalonado é multiplicado pela saída filtrada para se obter o estado escondido h_t a ser passado para a próxima célula: $h_t = O_t \times \tanh(C_t)$.

A arquitetura de uma rede LSTM está mostrada na Figura 2.

LSTM

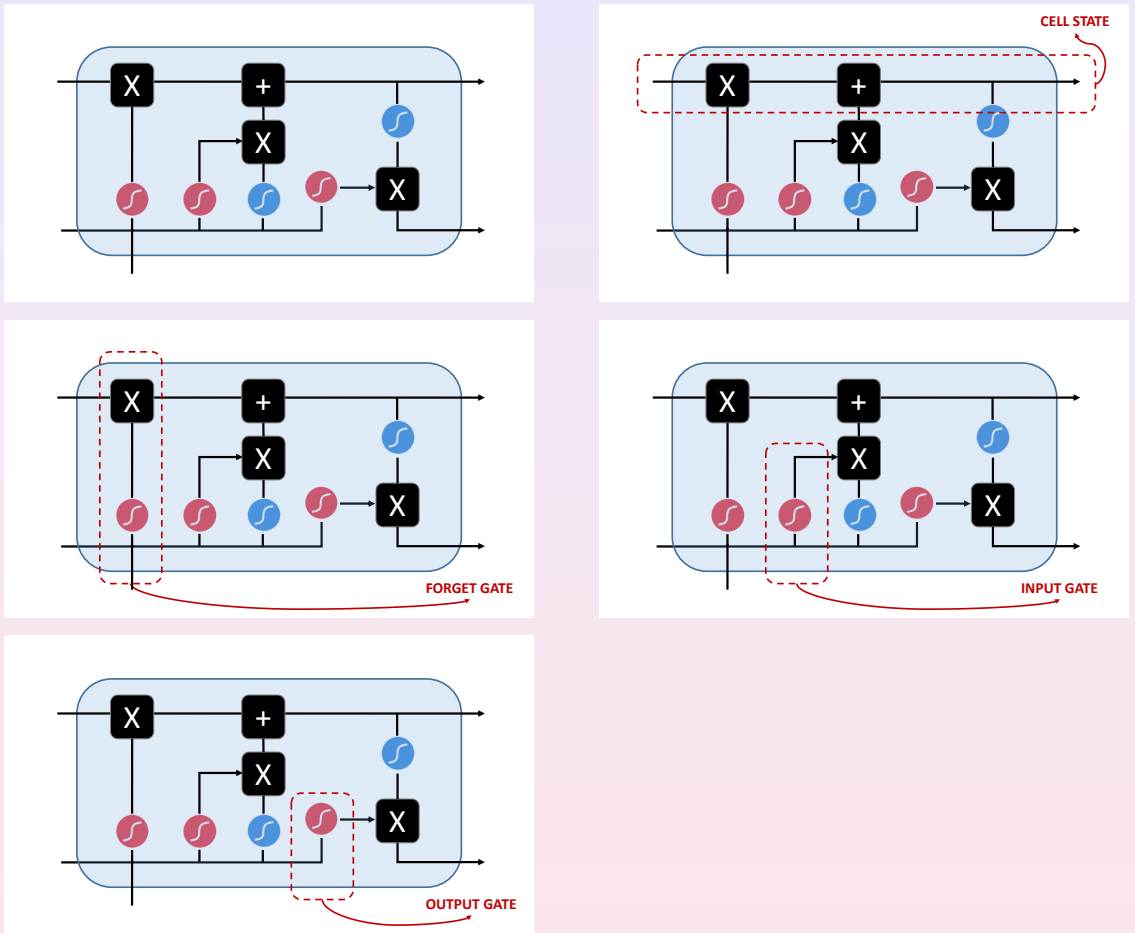


Figura 2: Arquitetura da célula de uma rede LSTM.

LSTM - Exemplo 1

- Consideremos os preços diários das ações da Petrobrás entre 31/08/1998 e 29/09/2010, totalizando $n = 2990$ observações, constantes do arquivo `acoes`.
- O gráfico da série está indicado na Figura 3, mostrando o seu caráter não estacionário.
- Como a rede LSTM funciona melhor para séries estacionárias ou mais próximas possíveis da estacionariedade, tomemos a série de primeiras diferenças, definida como $Y_t = X_t - X_{t-1}$, com X_t denotando a série original. A série de primeiras diferenças também está apresentada na Figura 3.
- Depois de obtidas as previsões deve-se fazer a transformação inversa para obter as previsões com a série original.
- Nosso intuito é fazer previsões para os dados de um conjunto de validação, com 897 observações, ajustando uma rede LSTM aos dados de um conjunto de treinamento, consistindo de 2093 observações.

LSTM - Exemplo 1

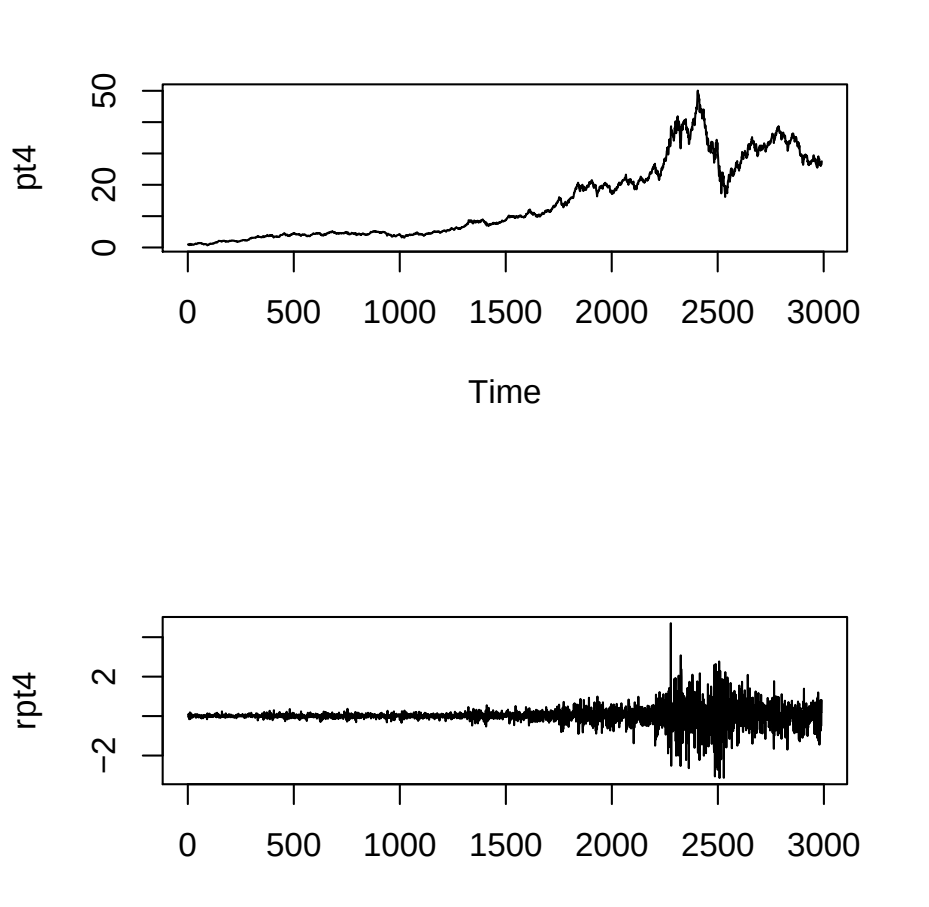


Figura 3: Série de preços das ações da Petrobrás e respectivas diferenças.

LSTM - Exemplo 1

- Como as redes LSTM supõem dados na forma de aprendizado supervisionado, ou seja, com uma variável resposta Y e uma variável preditora X, tomamos valores defasados da série de forma que os valores obtidos até o instante $t-k$ serão considerados como variáveis preditoras e o valor no instante t será a variável resposta. Neste exemplo, tomamos k=1.
- Para esse efeito, consideramos os comandos

```
lag_transform <-function(x, k=1){
lagged = c(rep(NA,k),x[1:(length(x)-k)])
DF = as.data.frame(cbind(lagged,x))
colnames(DF) <- c(paste0('x-',k),'x')
DF[is.na(DF)] <- 0
return(DF)
}
supervised = lag_transform(Petrobras4, 1)
head(supervised)
```

LSTM - Exemplo 1

- obtendo, para as 6 primeiras observações, a variável preditora na coluna `x-1` e a variável resposta, na coluna `x`.

```
    x-1     x
1  0.00  0.08
2  0.08  0.07
3  0.07 -0.06
4 -0.06 -0.04
5 -0.04 -0.04
6 -0.04 -0.03
```

- Para a divisão entre dados treinamento (70%) e de validação (30%), podemos utilizar os comandos

```
N <- nrow(supervised)
n <- round(N*0.7,digits=0)
train <- supervised[1:n,]
test <- supervised[(n+1):N,]
```

LSTM - Exemplo 1

- Em seguida, normalizamos os dados de entrada para que pertençam ao intervalo de variação da função de ativação, que é a sigmoide, com variação em [-1,1].
- Os valores mínimo e máximo do conjunto de treinamento são usados para normalizar os conjuntos de treinamento e de validação além dos valores preditos, por meio dos comandos

```
scale_data <- function(train, test, feature_range=c(0,1)){
  x<-train
  fr_min<-feature_range[1]
  fr_max<-feature_range[2]
  std_train<-((x-min(x))/(max(x)-min(x)))
  std_test<-(test-min(x))/(max(x)-min(x))
  scaled_train<-std_train*(fr_max-fr_min)+fr_min
  scaled_test<-std_test*(fr_max-fr_min)+fr_min
  return(list(scaled_train=as.vector(scaled_train),
  scaled_test=as.vector(scaled_test),
  scaler=c(min=min(x),max=max(x))))
}
```

LSTM - Exemplo 1

- continuando:

```
Scaled <- scale_data(train,test, c(-1,1))
y_train <- Scaled$scaled_train[,2]
x_train <- Scaled$scaled_train[,1]
y_test <- Scaled$scaled_test[,2]
x_test <-Scaled$scaled_test[,1]
```

- Para reverter os valores previstos à escala original, consideramos

```
invert_scaling <- function(scaled, scaler, feature_range=c(0,1)){
  min=scaler[1]
  max=scaler[2]
  t=length(scaled)
  mins=feature_range[1]
  maxs=feature_range[2]
  inverted_dfs=numeric(t)
  for(i in 1:t){
    X=(scaled[i]-mins)/(maxs-mins)
    rawValues=X*(max-min)+min
    inverted_dfs[i]<-rawValues
  }
  return(inverted_dfs)
}
```

LSTM - Exemplo 1

- A partir deste ponto iniciamos a modelagem. Com essa finalidade, precisamos fornecer o lote de entrada na forma de um vetor tridimensional, [`samples, timesteps, features`] a partir do estado atual [`samples, features`], em que `samples` é o número de observações em cada lote (tamanho do lote), `timesteps` é o número de passos para uma dada observação (para este exemplo, `timsteps`=1) e `features`=1, para o caso univariado como no exemplo.
- O tamanho do lote deve ser função dos tamanhos das amostras de treinamento e de validação. Usualmente esse valor é 1. Também devemos especificar `stateful = TRUE` de modo que após processar um lote de amostras os estados internos sejam reutilizados para as amostras do lote seguinte. Os comandos correspondentes são

```
dim(x_train) <- c(length(x_train), 1, 1)
X_shape2 <- dim(x_train)[2]
X_shape3 <- dim(x_train)[3]
batch_size=1
units=1
model <- keras_model_sequential()
model %>% layer_lstm(units,
    batch_input_shape = c(batch_size, X_shape2, X_shape3),
    stateful=TRUE) %>% layer_dense(units=1)
```

LSTM - Exemplo 1

- O modelo pode, então ser compilado, com a especificação do erro quadrático médio como função perda.
- O algoritmo de otimização é o *Adaptive Monument Estimation (ADAM)*. Usamos a acurácia como métrica para avaliar o desempenho do modelo.

```
model %>% compile(
  loss='mean_squared_error', optimizer=optimizer_adam(lr=0.02,decay=1e-6),
  metrics=c('accuracy'))
```

Por meio do comando `summary(model)` obtemos:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (1, 1)                    12
_________________________________________________________________
dense (Dense)                (1, 1)                    2
=================================================================
Total params: 14
Trainable params: 14
Non-trainable params: 0
_________________________________________________________________
```

LSTM - Exemplo 1

- Para ajustar o modelo, a rede LSTM exige o comando `shuffle=FALSE` cuja finalidade é evitar o embaralhamento do conjunto de treinamento e manter a dependência entre x_i e x_{i+t}.
- Além disso o algoritmo requer a redefinição do estado da rede após cada iteração (época).
- Os comandos para o ajuste do modelo são

```
Epochs=50
for(i in 1:Epochs){
  model %>% fit(x_train, y_train, epochs=1, batch_size=batch_size,
 verbose=1,shuffle=FALSE)
  model %>% reset_states()}
```

são geradas 50 épocas (iterações) da rede neural e em cada época é possível visualizar o valor da função perda e a acurácia.

LSTM - Exemplo 1

- Para fazer previsões usamos a função `predict()` e em seguida invertemos a escala e as diferenças para retornar à série original.

```
L=length(x_test)
scaler=Scaled$scaler
predictions=numeric(L)
for(i in 1:L){
  X=x_test[i]
  dim(X)=c(1,1,1)
  yhat=model %>% predict(X, batch_size=batch_size)
  yhat=invert_scaling(yhat, scaler, c(-1,1))
  yhat
  yhat=yhat+Pertrobras4[(n+i)]
  predictions[i] <- yhat}
```

- As previsões para as 897 observações do conjunto teste podem ser obtidas por meio da função `prediction()`. As 6 primeiras são apresentadas abaixo:

```
[1] -0.28833461  0.35166539 -0.16833461 -0.26833461  0.01166539
-0.03833461
```

LSTM - Exemplo 1

- As observações do conjunto de validação e as correspondentes previsões estão representadas na Figura 4.
- Na Figura 5 colocamos a série para o conjunto de treinamento (em azul), a série no conjunto de validação (em vermelho) e a série de previsões (em verde).
- A raiz quadrada do erro quadrático médio, calculado da maneira habitual, é $RMSE = 0,02835$.

LSTM - Exemplo 1

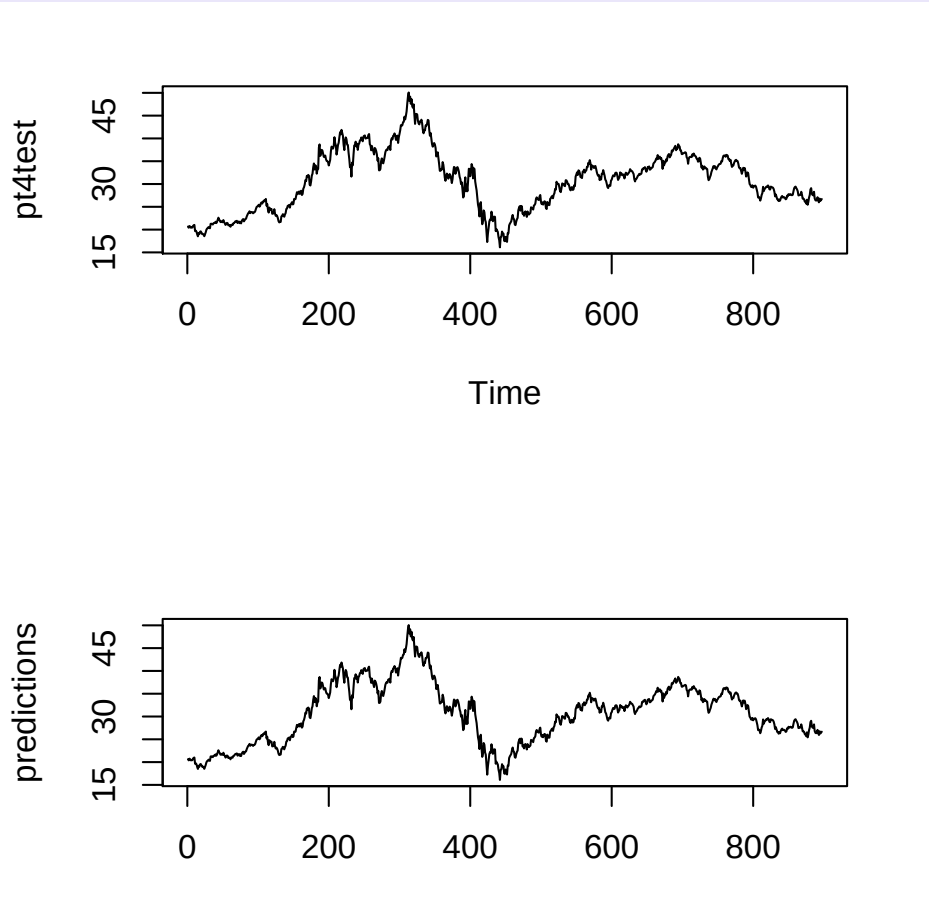


Figura 4: Série original e série de previsões de preços das ações da Petrobrás, no conjunto de validação.

LSTM - Exemplo 1

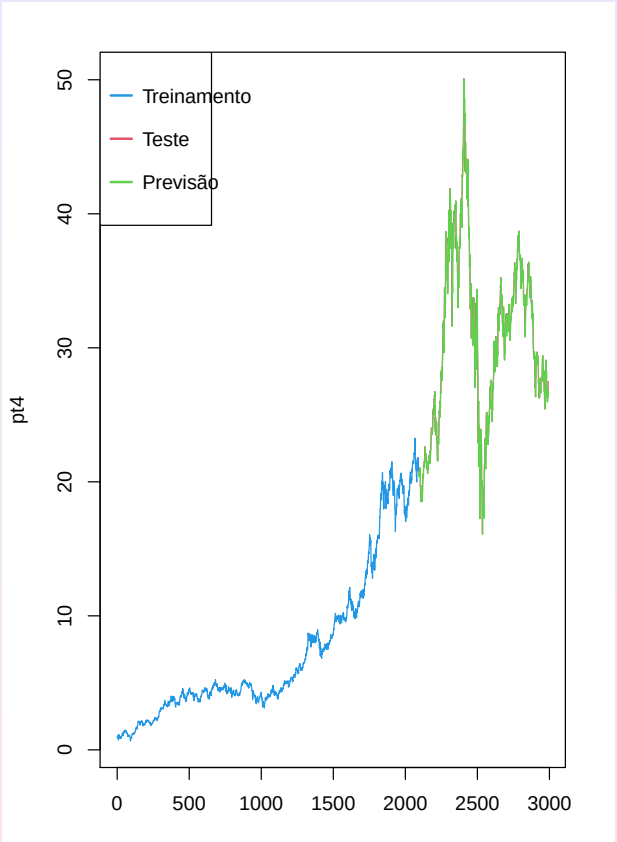


Figura 5: Série original de preços das ações da Petrobrás para o conjunto de treinamento (azul), para o conjunto de validação (vermelho) e série de previsões (verde).

LSTM - Exemplo 2

- Consideremos os dados do arquivo `covid2` com 526 observações de casos e óbitos por COVID-19 no Brasil entre 19/03/2020 e 25/08/2021.
- O gráfico da série está na Figura 6.
- Separemos $n = 369$ observações para o conjunto de treinamento e $m = 157$ observações para o conjunto de validação. O objetivo é prever o número de casos no conjunto de validação.
- Após uma análise por meio de uma rede LSTM similar àquela do Exemplo 1, o comando `summary()` produz o seguinte resultado:

LSTM - Exemplo 2

```
summary(model)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (1, 1)                    12
_________________________________________________________________
dense (Dense)                (1, 1)                    2
=================================================================
Total params: 14
Trainable params: 14
Non-trainable params: 0
_________________________________________________________________
```

LSTM - Exemplo 2

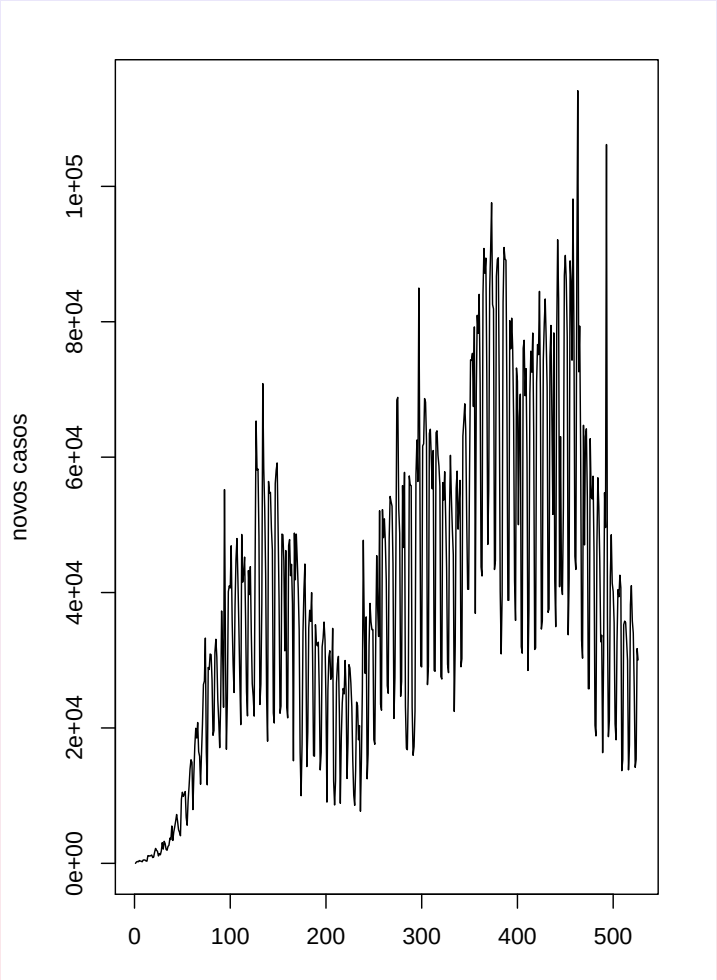


Figura 6: Série diária de novos infectados pelo vírus Covid-19 no Brasil.

LSTM - Exemplo 2

A raiz quadrada do erro quadráatico médio é $RMSE = 20793,87$ e as previsões podem ser vistas na Figura 7.

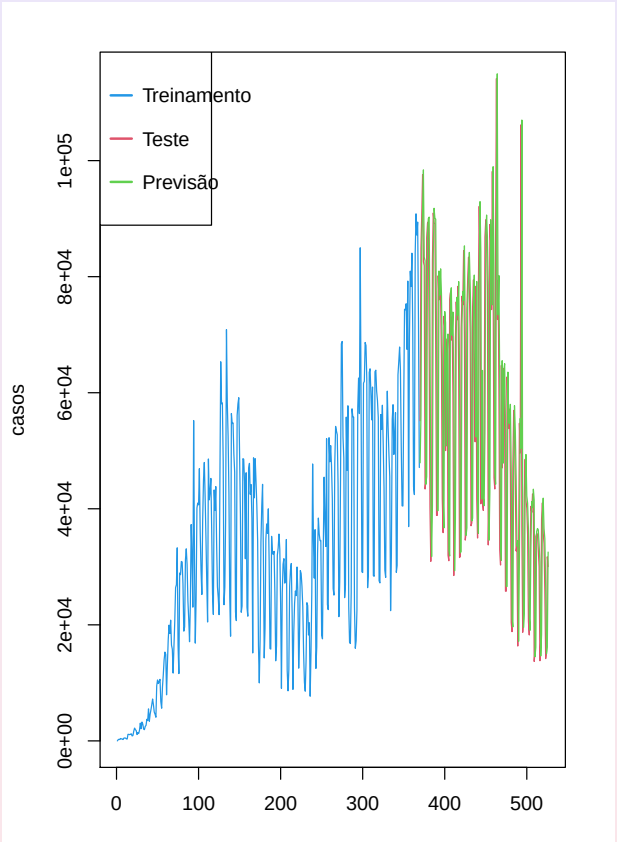


Figura 7: Série original de novos infectados por Covid-19 no Brasil para o conjunto de treinamento (azul), para o conjunto de validação (vermelho) e série de previsões (verde).

TORCH - Exemplo 3

- Consideremos os dados de novos casos de Covid-19 no Brasil de 25/02/2020 a 18/03/2023, total de $n = 1118$ observações diárias, constantes do arquivo `covidBrasil2023`. O gráfico da série está na Figura 8.
- Separemos $n = 782$ observações para o conjunto de treinamento, $m_1 = 168$ observações para o conjunto de validação e $m_2 = 168$ observações para o conjunto teste. O objetivo é prever o número de casos nos conjuntos de validação e teste.
- Neste exemplo vamos usar o pacote `Torch` juntamente com os pacotes `tidyverse, tidymodels, luz, timetk` e `cowplot`.
- Os valores da perda (loss), MAE (mean absolute error) e RMSE (root mean square error) obtidos após 150 épocas foram 6315, 6400 e 9622, respectivamente. A Figura 9 ilustra o decaimento dessas métricas ao longo das épocas.

TORCH - Exemplo 3

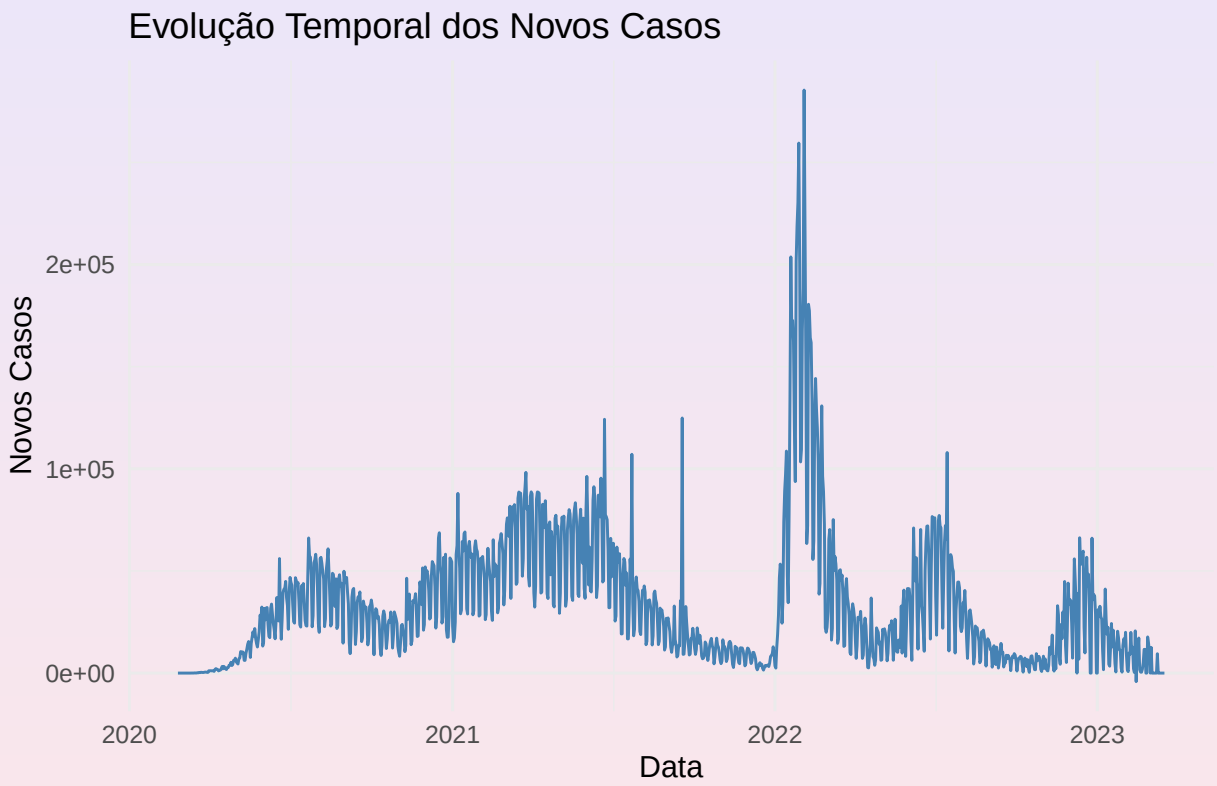


Figura 8: Série diária de novos infectados pelo vírus Covid-19 no Brasil.

TORCH - Exemplo 3

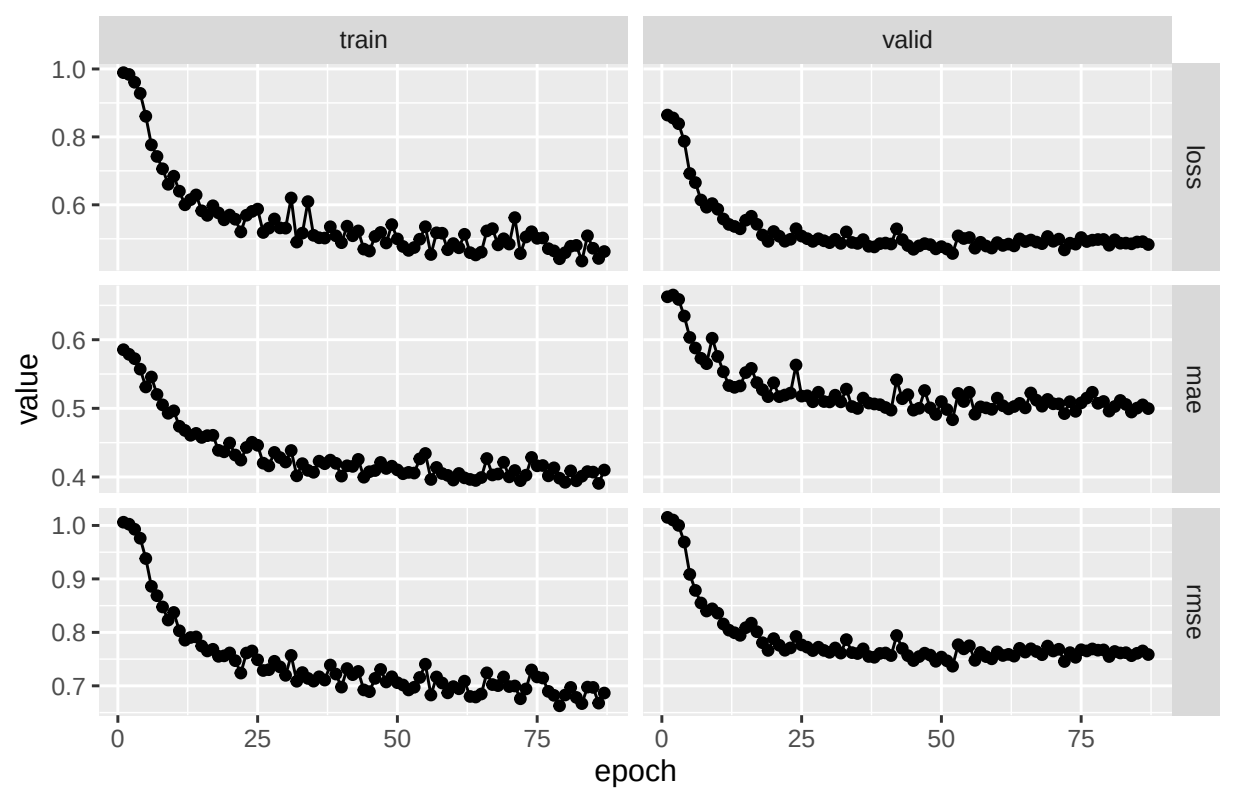


Figura 9: Métricas nos conjuntos de treinamento e validação.

TORCH - Exemplo 3

A Figura 10 mostra o conjunto de treinamento e as previsões nos conjuntos de validação e teste. A Figura 11 mostra essas previsões com mais detalhes.

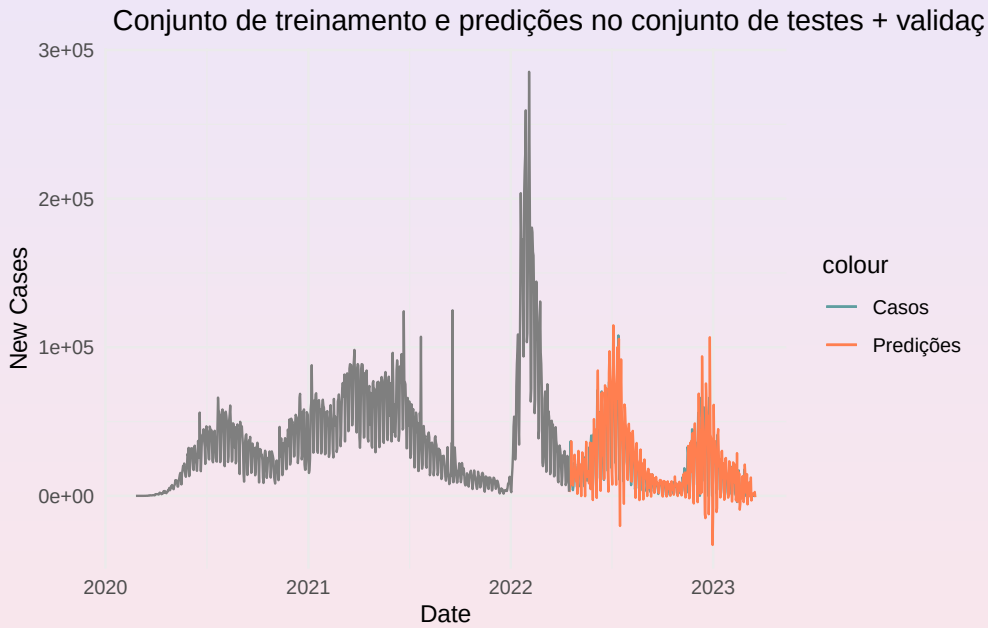


Figura 10: Conjunto de treinamento (preto) e previsões nos conjuntos de validação e teste (vermelho).

TORCH - Exemplo 3

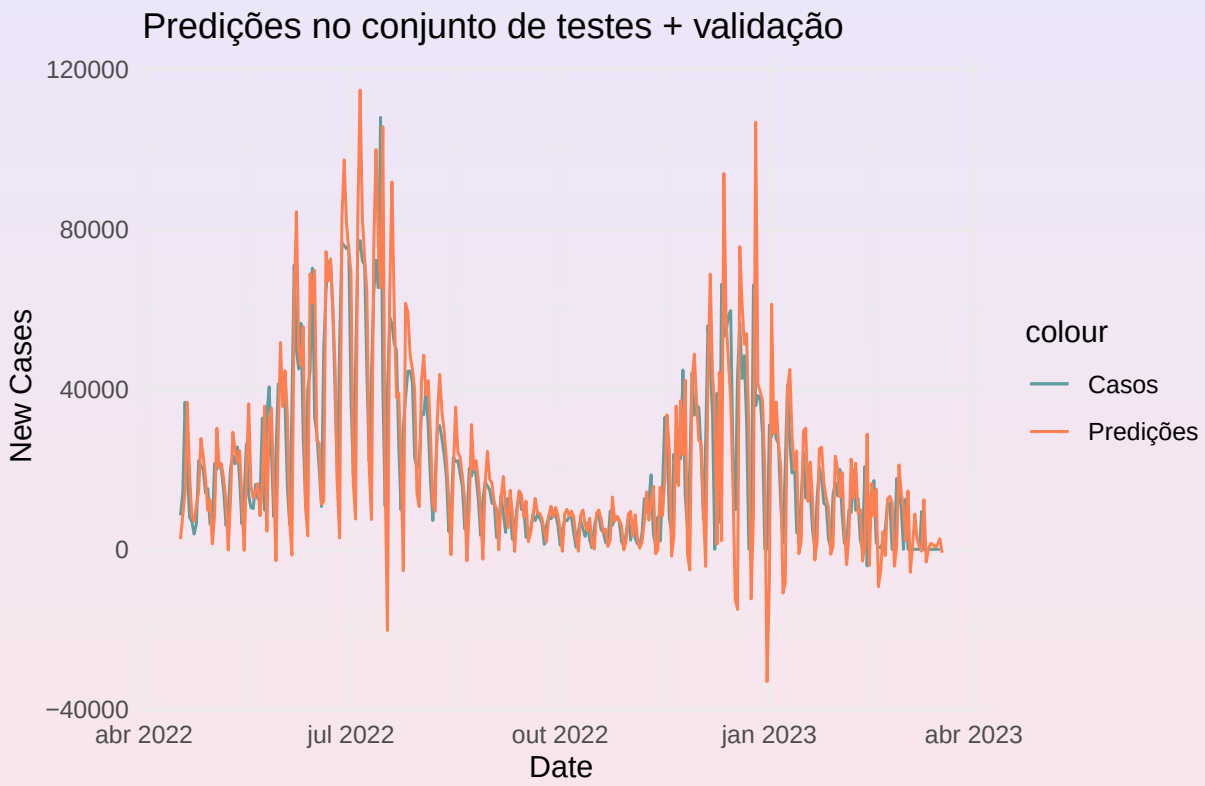


Figura 11: Previsões nos conjuntos de validação e teste mais detalhadas.

Referências

Chollet, F. (2018). *Deep Learning with R*. Manning.

Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*. The MIT Press.

Hastie, T., Tibshirani, R. and Friedman, J. (2017). *The Elements of Statistical Learning*, 2nd ed. New York: Springer.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning. *Nature*, **521**, 436–444.

Morettin, P. A. e Singer, J. M. (2024). *Estatística e Ciência de Dados*. Segunda edição. LTC: Rio de Janeiro.