Séries Temporais para Engenharia e Outras Áreas

Autores
GUSTAVO ROCHA SILVA
ORLANDO YESID ESPARZA ALBARRACIN
ROGÉRIO DE OLIVEIRA

Índice

Prefácio

Este livro se destina a você que está interessado em estudar e usar modelos de Análise e Previsão de Séries Temporais em campos como Engenharia (volume de vendas de produtos), Comércio (preços de artigos), Finanças (índices de ações), Ciências (valores de precipitação e temperatura) e, até mesmo, Ciências Sociais e da Saúde (indicadores sociais, casos de covid).

Ele foi elaborado como um material de apoio para disciplinas de cursos de Engenharia e Administração, mas pode ser empregado total ou parcialmente em cursos de diferentes áreas. Sendo seu objetivo principalmente didático, adotamos a forma dialógica para o texto e vamos nos referir muitas vezes diretamente ao leitor.

Ao longo do texto não só explicamos e discutimos os principais conceitos de Análise de Séries temporais, como também apresentamos exemplos, modelando dados que julgamos ser de interesse para um público diverso. Destacam-se os dados das Séries históricas de índices da Bolsa de Valores, da Temperatura Global e de demanda por aluguel de bicicletas. Para a implementação dos modelos empregamos a linguagem Python, contudo, buscamos disponibilizar um texto que possibilite a leitura para pessoas que não estejam familiarizadas com Python ou que não estejam interessadas em detalhes da codificação. A ideia é que você possa entender e acompanhar os resultados das análises, deixando os detalhes de programação para uma segunda leitura ou para quando você tiver a necessidade de aplicar esses modelos. Procuramos, assim, criar um texto de fácil compreensão tanto para programadores como para não programadores.

De qualquer modo, para quem quiser iniciar neste campo da programação e desenvolver seus próprios modelos, conceitos básicos como Introdução ao Pyhton, Pandas e Matplotlib são disponibilizados no site do livro, assim como tópicos avançados e todo o código implementado nesta obra.

Neste livro, você encontrará, além dos modelos tradicionais como AR, ARMA, ARIMA, ARIMAX etc., discutidos na maioria dos livros-textos, um capítulo dedicado ao tratamento de Séries Temporais com modelos de Aprendizado de

Máquina e Redes Neurais Profundas, fornecendo, assim, uma maior variedade de modelos atuais que podem ser implementados para analisar problemas que envolvem séries com grande volume de dados, complexidade e multidimensionalidade.

Em cada capítulo você vai encontrar ainda uma seção *Saiba Mais* com indicação de recursos ou referências sobre o tema estudado no capítulo.

Todos os códigos deste livro foram desenvolvidos e executados em Python no ambiente Google Colab e podem ser obtidos em https://github.com/*/Temporal. Lá você também encontrará atualizações do código, do texto e materiais complementares, bem como as bases de dados empregadas.

6. Aprendizado de Máquina

Neste capítulo você irá aprender:

1. O que é o Aprendizado de Máquina Supervisionado
2. A importância dos Modelos de Aprendizado de Máquina na previsão de Séries não Lineares e Multidimensionais
3. Como aplicar Modelos de Aprendizado Tradicionais com o Scikit-Learn e modelos de Deep Learning com o PyTorch

Modelos de Aprendizado de Máquina, ou ML (*Machine Learning*), são aplicados a uma série de tarefas que vão da identificação de fraudes em cartões de crédito e a recomendação de produtos, até sistemas para precificação e previsão das condições do tempo.

Esses modelos também vêm sendo aplicados com sucesso na previsão de Séries Temporais complexas, principalmente na presença de não linearidades e modelos multimodais. Como você aprendeu nos capítulos anteriores todos os modelos AR, MA, ARMA, ARIMA etc. são modelos de previsão lineares, no sentido de que os valores das séries são aditivos ou multiplicativos (veja no capítulo *Decomposição e Tratamento de Séries*). Os modelos de Aprendizado de Máquina não apresentam essa restrição, o que permite solucionar problemas bastante complexos na previsão de Séries Temporais. Eles ainda são bastante fáceis de usar pois aproximam bastante um grande conjunto de Séries com pouca ou quase nenhuma informação sobre sua estrutura (tendência, sazonalidade, periodicidade etc.) embora, essa falta de poder explicativo dos modelos de ML seja em muitos casos uma desvantagem desses modelos.

Embora este texto seja uma introdução ao tema de Séries Temporais, o uso recente desses modelos vem crescendo em vários seguimentos, da previsão de demanda de produtos à aplicações em ciência, telecomunicações e medicina, com soluções suportadas por grandes empresas de tecnologia como Meta (Prophet, Neuralprophet, PyTorch) e Google (TensorFlow). Assim, incluímos este capítulo para você conhecer esses modelos e algumas de suas

aplicações, mesmo sendo um conteúdo que você não irá encontrar nos textos mais tradicionais da área.

O Aprendizado de Máquina Supervisionado

O Aprendizado de Máquina tem uma série de variantes para a solução de diferentes problemas que vão da classificação e agrupamento de notícias e imagens, à detecção de anomalias no tráfego de internet ou a tradução automática de texto. Nesta introdução, para nossos propósitos, será suficiente nos determos nos modelos de **Aprendizado Supervisionado**. Esses modelos buscam encontrar, à partir de um conjunto de dados de entrada e saída, uma função que melhor ajusta esse conjunto de dados. O **Modelo de Regressão Linear**, que discutimos no capítulo *Decomposição e Tratamento de Séries*, é um modelo típico de Aprendizado Supervisionado e empregaremos ele para que você possa entender como esses modelos funcionam de modo geral.

Aproximando uma Função: O Preço de Imóveis

Você pode imaginar que o preço de um imóvel é certamente, dentre outros fatores, uma *função* de características do imóvel com área construída, número de suítes e garagens, ano da construção, sua localização etc. Colocamos *função* em destaque por quê, de fato, não conhecemos a *função real*, mas podemos obter um conjunto de dados de imóveis com suas características e preço e buscar um modelo ou função que melhor aproxima esses dados. Essa é a ideia por trás dos modelos supervisionados.

Nós não temos o preço de todos os imóveis, mas podemos obter uma amostra que, sendo significativamente grande, será útil para inferirmos sobre o preço de outros imóveis que não constam da amostra. Vamos chamar esse conjunto de **Conjunto de Treinamento**, e ele é a base dos modelos de aprendizado supervisionado.

Existem muitos modelos que podem ser empregados para aproximarem um conjunto de valores de entrada e suas saídas. Esses modelos podem ser agrupados em **Classes de Modelos**, como a Classe dos Modelos de Regressão Linear. Podemos escolher essa classe como nossa hipótese para

obter um modelo e aproximar o preço dos imóveis. Note, entretanto, que existem infinitos modelos que podem ser aplicados, para os infinitos valores de coeficientes da regressão:

$$\textbf{Preço Imóvel} = a_0 + a_1 \times \textbf{Área} + a_2 \times \textbf{nr Suítes} \ldots$$

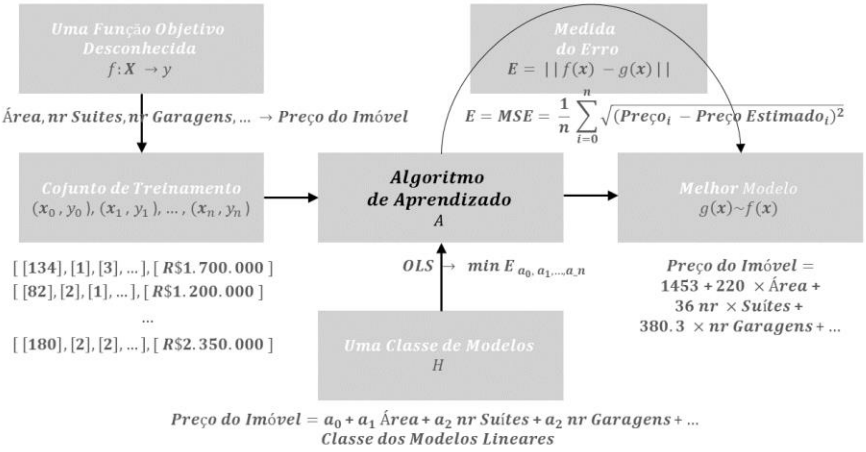


Fig. 1. Esquema geral do Aprendizado Supervisionado de Máquina e um exemplo de aplicação à Previsão de Preços de Imóveis por Modelos Lineares.

Para encontrarmos o melhor modelo, dentre todos os modelos lineares possíveis, precisamos estabelecer uma métrica. Afinal, o que é o melhor modelo? Podemos, por exemplo, buscar o modelo que melhor aproxima os valores obtidos aos valores do nosso conjunto de treinamento empregando para isso o erro quadrático médio, **MSE** (e, a depender da classe de modelo, muitas outras métricas poderiam ser empregadas para definir o *melhor*).

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(\text{Preço}_i - \text{Preço Estimado}_i)^2$$

E o aprendizado pode então se resumir a um problema de otimização de encontrar os valores dos coeficientes $a_0, a_1, \ldots, a_n$ que minimizam o erro,

$$\min_{a_0, a_1, \ldots, a_n} MSE = \frac{1}{n}\sum_{i=0}^{n}(\text{Preço}_i - \text{Preço Estimado}_i)^2$$

O que é em geral, no caso da regressão linear, obtido por um método de mínimos quadrados (OLS).

Outros modelos

Mas no aprendizado de máquina muitas outras classes de modelos, além dos modelos lineares, podem ser aplicados.

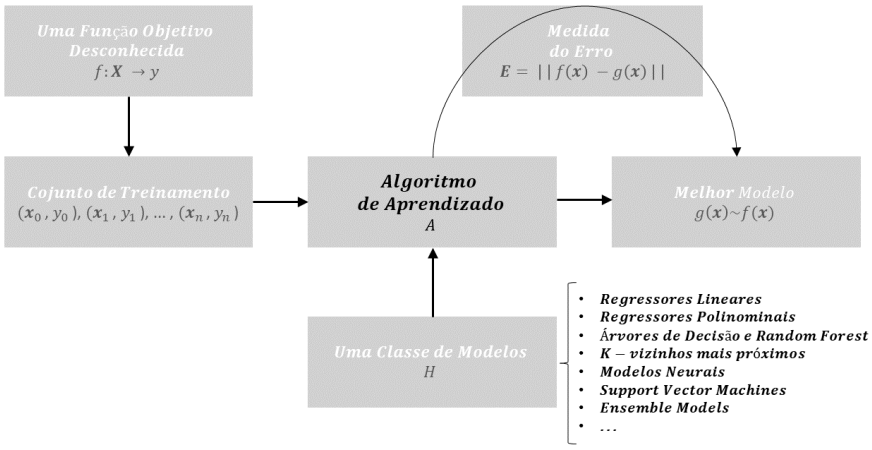


Fig. 2. Esquema geral do Aprendizado Supervisionado de Máquina e outras Classes de Regressores.

Esses modelos empregam outros princípios que podem divergir bastante e levar a resultados bastante diferentes dos modelos lineares. Você pode pensar, por exemplo, nos modelos regressão polinomial ou de aproximação de funções por séries, como as séries de Taylor e Fourier. Mas ainda existem modelos que segue paradigmas bastante diferentes desses modelos, como modelos baseados em Árvores de Decisão, K-Vizinhos mais Próximos ou os modelos de Redes Neurais Artificiais.

Não vamos entrar aqui em detalhes de como cada um desses modelos funciona, mas o importante é que você entenda que, à exemplo do modelo de Regressão Linear, o Aprendizado de Máquina consiste em selecionar de uma classe de modelos o que melhor se ajusta aos dados, e que cada uma dessas classes pode ter uma forma bastante diferente do método que empregamos acima, embora a ideia seja sempre a mesma.

Aplicando Diferentes Modelos

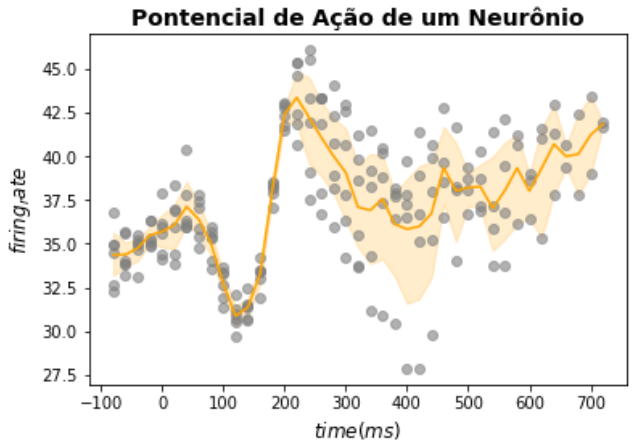
Vamos então ver como podemos aproximar uma função por outras classes de modelos mais comuns ao Aprendizado de Máquina, nos limitando aqui aos modelos **regressores**, isto é, modelos que estimam valores em contraposição aos modelos de classificação, também bastante comuns, mas que não são de interesse para nós no momento. Empregaremos vários modelos para aproximar agora uma função temporal conhecida como Potencial de Ação.

O Potencial de Ação é uma *função* que explica a ativação dos neurônios e que tem um papel importante nas neurociências e aplicações médicas. Colocamos *função* em destaque por quê, de fato, não conhecemos a *função real* desse potencial, mas podemos obter uma série de medidas no tempo e buscar aproximar esses dados por algum modelo ou função.

```
import seaborn as sns
df = sns.load_dataset('dots')
df.head()
```

```
   time  firing_rate
0   -80    34.970107
1   -80    36.785815
2   -80    34.478506
3   -80    34.991424
4   -80    32.241533
```

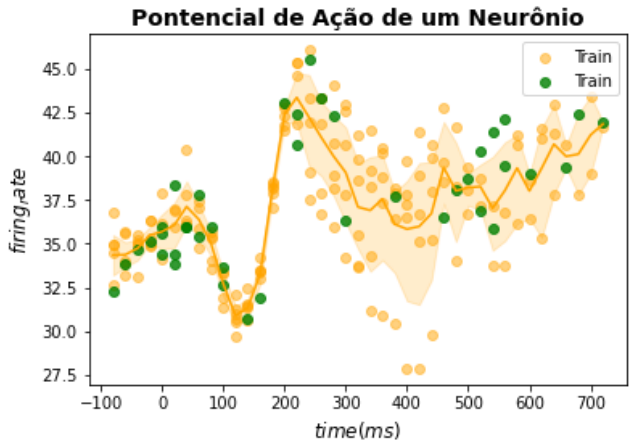
```
sns.lineplot(x=df.time, y=df.firing_rate,color='orange')
plt.scatter(x=df.time, y=df.firing_rate,color='gray',alpha=0.6)
```



A linha amarela representa uma suavização do conjunto de pontos da série à exemplo do que você viu no capítulo *Modelos de Previsão por Suavização*. Vamos separar nesse conjunto de dados os dados de **Treinamento** e **Teste**

fizemos nos capítulos anteriores na construção dos modelos lineares. A diferença aqui é que, por hora, sendo uma função do tempo esses conjuntos não precisam ser contíguos. Faremos isso mais adiante quando tratarmos Séries Temporais com dependência sequencial (capítulo *Decomposição e Tratamento de Séries*).

```
from sklearn.model_selection import train_test_split

X = df[['time']]
y = df.firing_rate

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1)

sns.lineplot(x=df.time, y=df.firing_rate,color='orange')
plt.scatter(x=X_train.time,y=y_train,color='orange',alpha=0.5,label='T
rain')
plt.scatter(x=X_test.time,y=y_test,color='green',alpha=0.8,label='Trai
n')
```



Vamos começar com um modelo de Regressão Linear simples. Assim você poderá entender como são aplicados modelos Regressores em geral com o Scikit-Learn, a principal biblioteca de ML para modelos não profundos (Deep Learning).

De modo geral os regressores (chamados também de estimadores) são aplicados do seguinte modo:

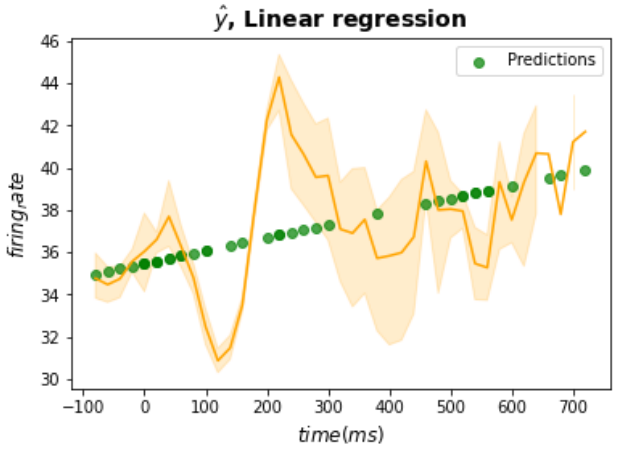
```
# instancia o estimador, a classe de modelos a ser empregada
  regressor = <regressor>(<regressor_parameters>)

# treina o modelo, encontrando o melhor modelo da classe
  regressor.fit(X_train, y_train)

# Faz a previsão dos valores de teste
  y_pred = regressor.predict(X_test)
```

Fig. 3. Esquema geral de Programação de Estimadores de Aprendizado.

Modelo Linear

Abaixo você encontra uma aplicação do modelo linear para estimar a função de Ativação do nosso exemplo.

```
from sklearn import linear_model

regressor = linear_model.LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

sns.lineplot(x=X_train.time,y=y_train,color='orange')
plt.scatter(X_test,y_pred,color='green',marker='o',alpha=0.7,label='Predictions')
```

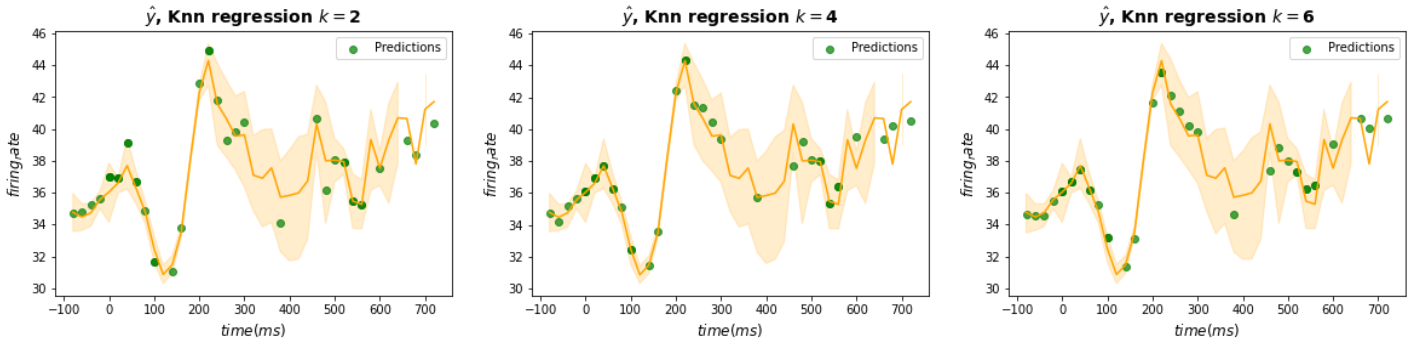


Esse é o melhor modelo que a classe de regressores lineares pode fornecer e, de fato, você observa que ele se ajusta muito pouco aos dados. Modelos lineares não podem capturar a não linearidade da função de ativação.

Modelo Regressor K-Vizinhos mais Próximos

O modelo **K-Vizinhos mais Próximos** é um modelo que ilustra bastante bem haverem outros princípios que permitem a aproximação de funções. A ideia do K-Vizinhos mais Próximos é que a estimativa de um valor desconhecido da função, ou do conjunto de teste, pode ser obtido por uma média do valor dos vizinhos mais próximos no conjunto de treinamento. O número de vizinhos k variando de 1 a n elementos do conjunto mais, em geral, não sendo muito superior a uma ou duas dezenas.

```
from sklearn.neighbors import KNeighborsRegressor

for k in range(1,4):

  regressor = KNeighborsRegressor(n_neighbors=2*k)
  regressor.fit(X_train, y_train)

  y_pred = regressor.predict(X_test)

  plt.subplot(1,3,k)
  sns.lineplot(x=X_train.time,y=y_train,color='orange')

plt.scatter(X_test,y_pred,color='green',marker='o',alpha=0.7,label='Predictions')
```



É uma ideia bastante simples e basta você encontra os k vizinhos mais próximos de um ponto que você deseja estimar a função e fazer a média desses valores. Apesar da simplicidade o modelo consegue capturar bastante

bem a não linearidade da função como você pode ver acima. Esse modelo, entretanto, não é adequado para *forecasting*, isto é as estimativas de valores *futuros* e fora do intervalo de valores do conjunto de treinamento e veremos mais adiante como ajustar esses modelos para a previsão de séries temporais.

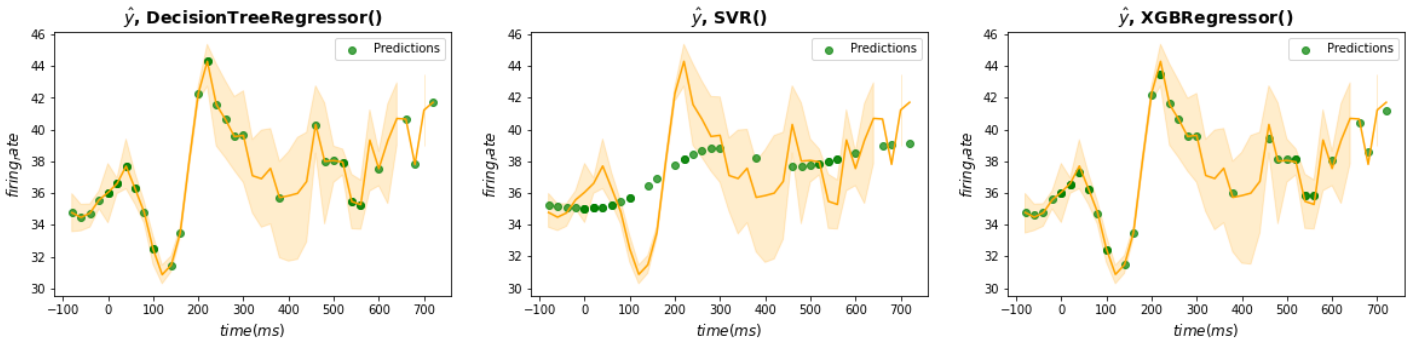
Outros Modelos Regressores

Existem muitos outros modelos e que empregam paradigmas bastante diferentes do modelo Linear ou do modelo de k-Vizinhos mais próximos e você poderá encontrar muitas referências sobre isso. Deixamos ao final algumas na seção final *Saiba Mais* deste capítulo.

O que nos interessa é saber que existem muitos modelos e que, por partirem de princípios de construção diferentes, levam a diferentes soluções de aproximação da função original.

Abaixo você pode ver a aplicação de 3 modelos regressores bastante empregados o **Decision Tree Regressor**, um modelo baseado em **Máquinas de Vetores de Suporte** e **XGBoosting Regressor**. Todos têm a mesma lógica de aplicação dos modelos anteriores, linear a k-vizinhos mais próximos, e você pode alterar o código abaixo para experimentar ainda outros modelos.

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from xgboost.sklearn import XGBRegressor
from sklearn.linear_model import BayesianRidge
from sklearn.linear_model import ElasticNet
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import SGDRegressor
from lightgbm import LGBMRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor

base_estimators = [ DecisionTreeRegressor(),SVR(), XGBRegressor() ]

plt.figure(figsize=(20,4))
k = 1
for regressor in base_estimators:
  regressor.fit(X_train, y_train)
  y_pred = regressor.predict(X_test)
  plt.subplot(1,3,k)
  k = k + 1
  sns.lineplot(x=X_train.time,y=y_train,color='orange')
  plt.scatter(X_test,y_pred,color='green',marker='o',alpha=0.7)
```



Visualmente o SVR não apresenta uma boa aproximação, mas vamos lembrar que neste exemplo simples não exploramos todos os parâmetros que podemos aplicar a esses modelos.

O Aprendizado de Máquina é bastante mais amplo do que aprensentamos aqui. Aqui nos limitamos a apresentar uns poucos exemplos de Aprendizado de Máquina Supervisionado unicamente para tarefas de Regressão (estimativa de valores) e se você tiver interesse neste tema sugerimos vistar algumas das referências que indicadas no final desse capítulo.

Essa introdução, de qualquer modo, será suficiente para aplicarmos modelos de ML para a previsão de Séries Temporais complexas e não lineares.

Bike Sharing: Machine Learning

No capítulo anterior você pôde acompanhar a análise completa de uma Série de Dados bastante complexa. O conjunto de dados Bike Sharing é uma Série bastante complexa embora com características bastante comuns em um grande número de problemas que envolvem demanda de produtos e serviços. Você pôde acompanhar uma série transformações (time index, resample etc.) que aprendemos e de análises (autocorrelação, *detrending*, autocorrelação parcial etc.) e que permitiram a construção de um modelo ARIMA de previsão. Reproduzimos aqui os pontos essenciais para podermos aplicar, neste mesmo conjunto de dados, modelos de Aprendizado de Máquina.

Preparação dos Dados

Nosso arquivo traz dados hora a hora da demanda do aluguel de bicicletas e, após a indexação dos dados, vamos fazer um *resample* para obter os valores

máximos por dia. A ideia é a previsão dos valores máximos para que não haja falta da oferta do serviço.

```
df = pd.read_csv('https://github.com/…/london_merged.csv')
df.head()
```

```
              timestamp  cnt   t1   t2    hum  wind_speed  weather_code  \
0  2015-01-04 00:00:00  182  3.0  2.0   93.0         6.0           3.0
1  2015-01-04 01:00:00  138  3.0  2.5   93.0         5.0           1.0
2  2015-01-04 02:00:00  134  2.5  2.5   96.5         0.0           1.0
3  2015-01-04 03:00:00   72  2.0  2.0  100.0         0.0           1.0
4  2015-01-04 04:00:00   47  2.0  0.0   93.0         6.5           1.0
```

```
from datetime import datetime
df.timestamp = pd.to_datetime(df.timestamp)
df['year'] = df.timestamp.dt.year
df = df[df.year == 2015]
df = df.set_index('timestamp')
```

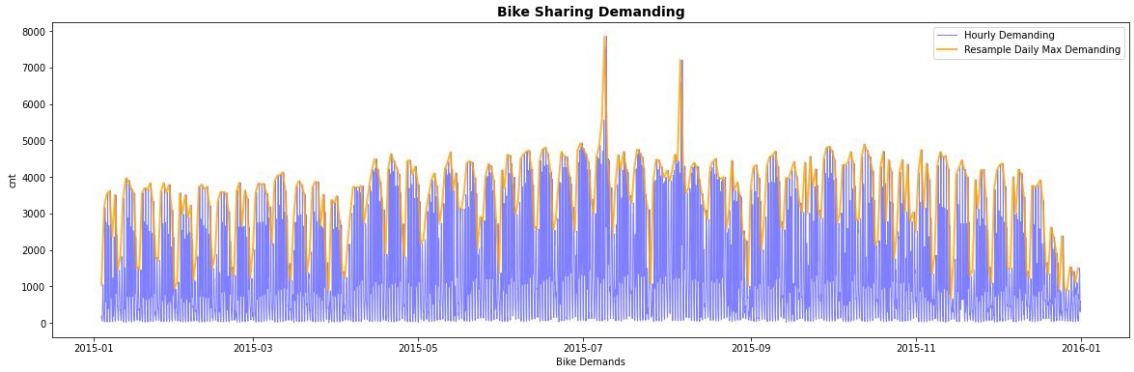
```
df_resample = df.copy()
df_resample = df_resample.resample('D').max()
```

```
df_resample.head()
```

```
             cnt    t1    t2    hum  wind_speed  weather_code  is_holiday  \
timestamp
2015-01-04  1039   4.0   3.5  100.0        15.0           4.0         0.0
2015-01-05  3161  10.0   9.0   93.0        14.0           4.0         0.0
2015-01-06  3534  11.0  11.0   88.0        27.5           7.0         0.0
2015-01-07  3618  10.0   7.0   87.0        30.5           7.0         0.0
2015-01-08  2351  12.0  12.0   88.0        27.0           7.0         0.0
```

A série de dados que temos interesse é agora a dos valores cnt, valores máximos de demanda de bicicletas por dia e representada pela linha amarela no gráfico abaixo.

```
plt.figure(figsize=(20,6))

sns.lineplot(x = df.cnt.index, y = df.cnt, color='blue', alpha=0.5,
label='Hourly Demanding',lw=1)
sns.lineplot(x = df_resample.cnt.index, y = df_resample.cnt,
color='orange', alpha=0.8, label='Resample Daily Max Demanding',lw=2)
```



Modelo ARIMA

Depois das análises de autocorrelação e testes de tendência (ver capítulo Modelos de Previsão ARMA e ARIMA) chegamos ao seguinte modelo ARIMA ótimo fazendo o *forecasting* para os 20% dados finais da Série (Conjunto de Teste):

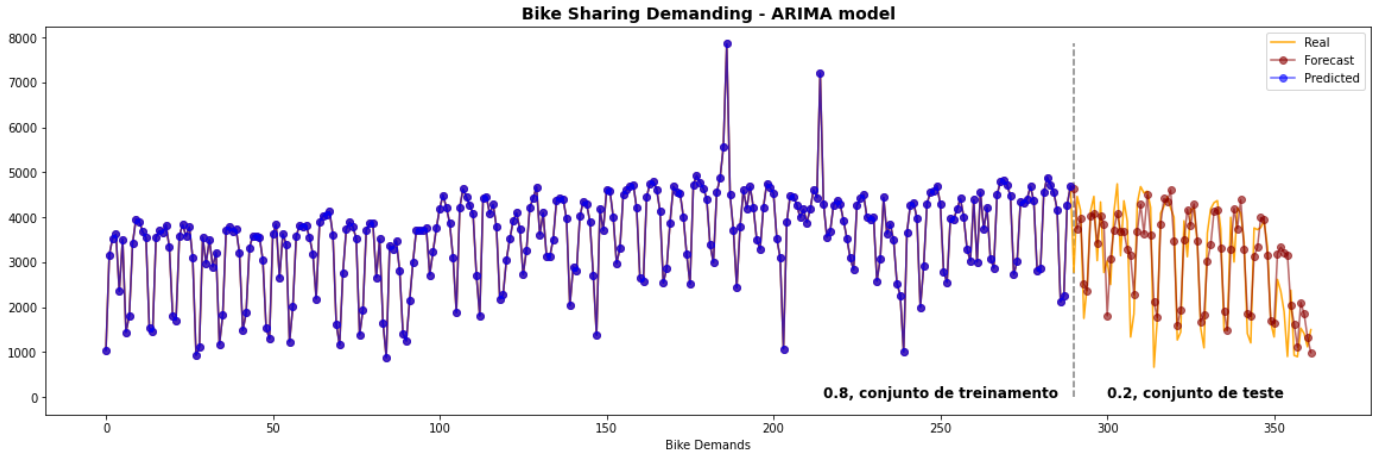
```
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.stattools import acf, pacf

df = df_resample[['cnt']]
df.columns = ['value']
df = df.reset_index(drop=True)
# df.head()

test_size = int(0.2*len(df))
train_size = len(df) - test_size

# Create Training and Test
train = df.value[:train_size]
test = df.value[train_size:]

p = 7
d = 1
q = 1

model = ARIMA(train, order=(p,d,q))
model_fit = model.fit()

print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.value   No. Observations:                  289
Model:                 ARIMA(7, 1, 1)   Log Likelihood               -2286.024
Method:                       css-mle   S.D. of innovations            654.251
Date:                Sun, 06 Mar 2022   AIC                           4592.048
Time:                        20:23:52   BIC                           4628.713
Sample:                             1   HQIC                          4606.740
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const            4.3920     10.093      0.435      0.664     -15.391      24.175
ar.L1.D.value   -0.1158      0.149     -0.778      0.437      -0.408       0.176
ar.L2.D.value   -0.3389      0.101     -3.344      0.001      -0.537      -0.140
ar.L3.D.value   -0.2527      0.112     -2.247      0.025      -0.473      -0.032
ar.L4.D.value   -0.2999      0.107     -2.814      0.005      -0.509      -0.091
ar.L5.D.value   -0.2907      0.109     -2.656      0.008      -0.505      -0.076
ar.L6.D.value   -0.1000      0.111     -0.902      0.368      -0.317       0.117
ar.L7.D.value    0.3738      0.084      4.455      0.000       0.209       0.538
ma.L1.D.value   -0.4729      0.148     -3.201      0.002      -0.762      -0.183
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -1.0494           -0.5174j            1.1700           -0.4271
AR.2           -1.0494           +0.5174j            1.1700            0.4271
AR.3           -0.2500           -1.0520j            1.0813           -0.2871
AR.4           -0.2500           +1.0520j            1.0813            0.2871
AR.5            0.6397           -0.8025j            1.0263           -0.1429
AR.6            0.6397           +0.8025j            1.0263            0.1429
AR.7            1.5867           -0.0000j            1.5867           -0.0000
MA.1            2.1147           +0.0000j            2.1147            0.0000
-----------------------------------------------------------------------------
```

```
history = [x for x in train]
predictions = list()

# forward validation
for t in test.index:
  # print(t)
      model = ARIMA(history, order=(7,2,1))
      model_fit = model.fit( )
      output = model_fit.forecast()
      yhat = output[0]
      predictions.append(yhat)
      obs = test[t]
      history.append(obs)

# metrics
from statsmodels.tsa.stattools import acf

def forecast_accuracy(forecast, actual, print_flag=True):
    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))  # MAPE
    me = np.mean(forecast - actual)                           # ME
    mae = np.mean(np.abs(forecast - actual))                  # MAE
    mpe = np.mean((forecast - actual)/actual)                 # MPE
    mse = np.mean((forecast - actual)**2)                     # MSE
    rmse = np.mean((forecast - actual)**2)**.5                # RMSE
    corr = np.corrcoef(forecast, actual)[0,1]                 # CORR
    acf1 = acf(forecast - actual)[1]                          # ACF1

    metrics = {'MSE':mse, 'MAPE':mape, 'ME':me, 'MAE': mae, 'MPE':
mpe, 'RMSE':rmse, 'ACF1':acf1, 'CORR':corr}

    if print_flag:
      for key, value in metrics.items():
```

```
        print(f'{key}: \t {value:.2f}')
    return metrics


y_pred = np.concatenate( (np.array( history[0:train_size] ).reshape(-
1), np.array(predictions).reshape(-1)) )

plot_forecast(pd.DataFrame( history ), y_pred, np.array(
history[0:train_size] ), title='Bike Sharing Demanding - ARIMA model')
cur_metrics = forecast_accuracy(y_pred[train_size::],
df[train_size::].values.reshape(-1))

dict_metrics = {}
dict_metrics['ARIMA Model'] = cur_metrics
```




```
MSE:                479701.79
MAPE:               0.27
ME:                 109.29
MAE:                535.00
MPE:                0.16
RMSE:               692.61
ACF1:               -0.12
CORR:               0.84
```

Aprendizado de Máquina

Como você acompanhou no começo desse capítulo os modelos de Aprendizado Supervisionado procuram *mapear* um conjunto de dados de entrada e suas saídas. Essa é a *função* que desejamos aproximar. A ideia é a mesma que você viu no capítulo *Decomposição e Tratamento de Séries* sobre Séries com dependência sequencial.

$$\begin{aligned}
&\text{Lag 1:} && x_{t-1} \rightarrow x_t \\
&\text{Lag 2:} && x_{t-2}, x_{t-1} \rightarrow x_t \\
&\text{Lag 3:} && x_{t-3}, x_{t-2}, x_{t-1} \rightarrow x_t \\
&\ldots \\
&\text{Lag n:} && x_{t-n}, x_{t-n-1}, x_{t-1} \rightarrow x_t
\end{aligned}$$

Podemos, assim, construir uma tabela a partir da série de dados original $\boldsymbol{X} \rightarrow y$ onde $\boldsymbol{X}$ é o conjunto de dados das n lags que queremos empregar em nosso modelo e y a nossa variável objetivo no tempo t.

A função abaixo constrói essa tabela para uma coluna de dados df e um número de *lags* n_lags. O parâmetro prefix é empregado para nomear as variáveis e será útil no tratamento de Séries Multidimensionais como as que estudamos no capítulo *Modelos SARIMAX e Multidimensionais*.

```
def ml_timeseries_df(df, n_lags, prefix=''):

      df_ml = pd.DataFrame()

      for i in range(n_lags,0,-1):
            df_ml[prefix + 't-' + str(i)] = df.shift(i).values[:,0]
      df_ml['t'] = df.values[:,0]
      df_ml = df_ml[n_lags:]
      df_ml = df_ml.reset_index(drop=True)

      return df_ml

df = ml_timeseries_df(df_resample[['cnt']], n_lags=7)
df.head()
```

	t-7	t-6	t-5	t-4	t-3	t-2	t-1	t
0	1039.0	3161.0	3534.0	3618.0	2351.0	3505.0	1436.0	1813
1	3161.0	3534.0	3618.0	2351.0	3505.0	1436.0	1813.0	3418
2	3534.0	3618.0	2351.0	3505.0	1436.0	1813.0	3418.0	3960
3	3618.0	2351.0	3505.0	1436.0	1813.0	3418.0	3960.0	3904
4	2351.0	3505.0	1436.0	1813.0	3418.0	3960.0	3904.0	3673

Fig. 4. Reconstrução da Série de Dados em uma Tabela de Entradas e Saídas. Os valores de saída estimados a partir de 7 valores anteriores da série (7-lags).

Organizado os dados desse modo podemos separar a primeira porção dos dados, 0.8, para estimar algum dos regressores de Aprendizado de Máquina que vimos antes e aplicarmos o *forecasting* ao restante, 0.2,dos dados da série (veja a Validação Cruzado no capítulo *Modelos de Previsão ARMA e ARIMA*).

```
def split_train_test(X, y, test_perc=0.2):

  test_size = int(len(df) * test_perc)
  train_size = len(df) - test_size

  X_train, y_train  = X[:train_size], y[:train_size]
  X_test, y_test = X[train_size:], y[train_size:]

  return X_train, y_train, X_test, y_test

X = df.drop(columns='t')
y = df[['t']]

X_train, y_train, X_test, y_test = split_train_test(X, y,
test_perc=0.2)
```

O código para aplicação do regressor é o mesmo que empregamos antes no exemplo da Função neuronal de Ativação, apenas que agora os dados de entrada e saída são outros.

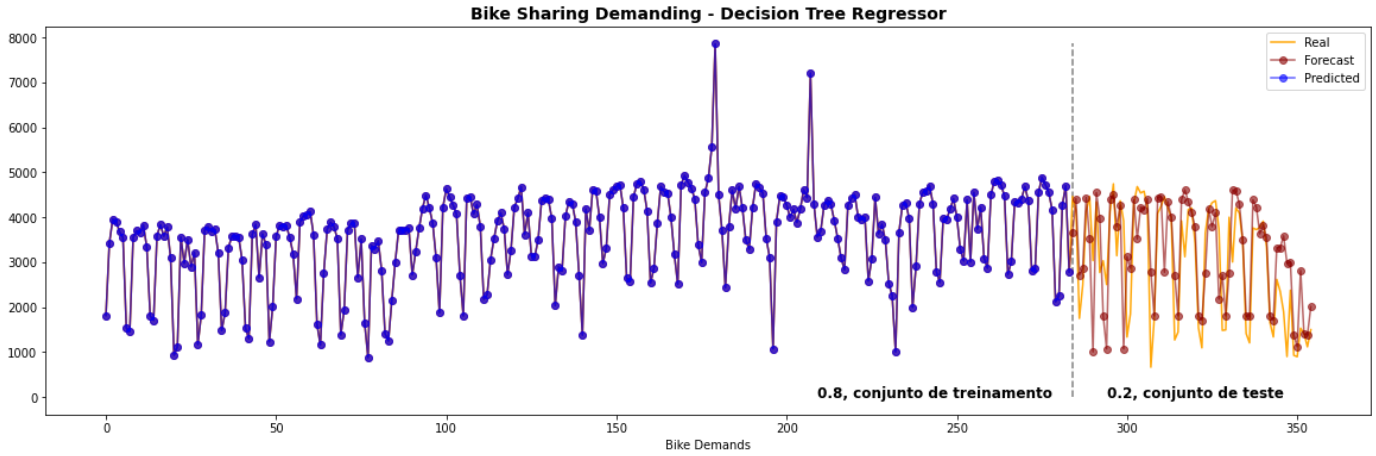
Decision Tree Regressor

Vamos primeiramente aplicar o regressor Decision Tree Regressor e verificar os resultados do modelo.

```
from sklearn.tree import DecisionTreeRegressor
```

```
regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X)

plot_forecast(y, y_pred, X_train, title='Bike Sharing Demanding -
Decision Tree Regressor')
```

```
cur_metrics = forecast_accuracy(y_pred[len(y_train):],
y[len(y_train):].values.reshape(-1))
dict_metrics['DecisionTreeRegressor'] = cur_metrics

MSE:          865131.56
MAPE:         0.34
ME:           180.15
MAE:          687.96
MPE:          0.20
RMSE:         930.12
ACF1:         0.07
CORR:         0.72
```

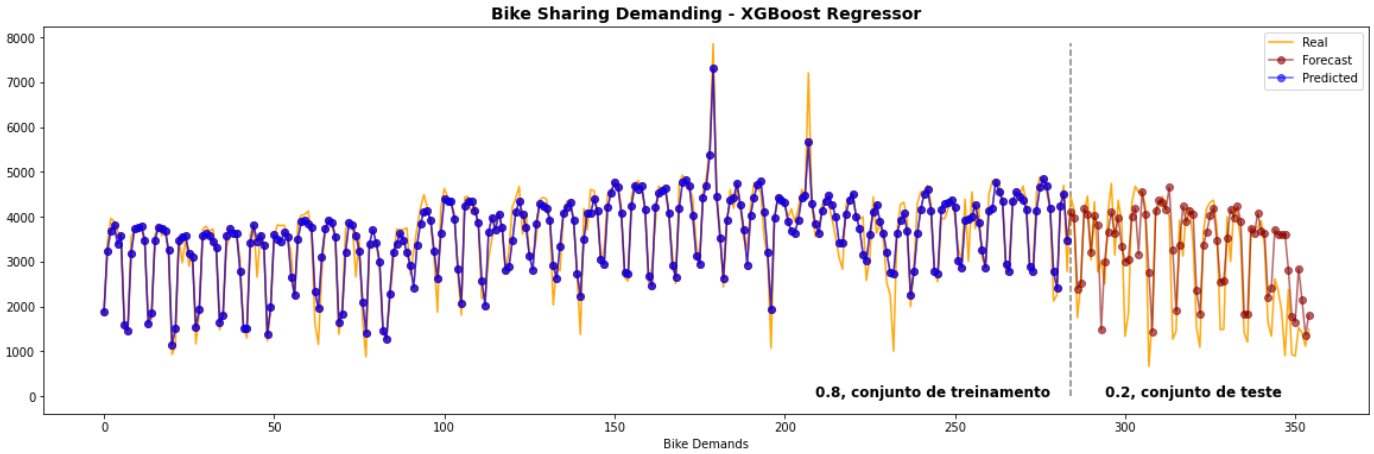


XGBoost Regressor

Do mesmo modo podemos aplicar o regressor XGBoost Regressor você pode experimentar outros modelos se quiser.

```
from xgboost.sklearn import XGBRegressor

regressor =  XGBRegressor(random_state=0)

regressor.fit(X_train,y_train.values.reshape(-1))
y_pred = regressor.predict(X)
```

```
plot_forecast(y, y_pred, X_train, title='Bike Sharing Demanding -
XGBoost Regressor')

cur_metrics = forecast_accuracy(y_pred[len(y_train):],
y[len(y_train):].values.reshape(-1))
dict_metrics['GradientBoostingRegressor'] = cur_metrics
```




```
MSE:          686190.71
MAPE:         0.36
ME:           307.29
MAE:          622.81
MPE:          0.28
RMSE:         828.37
ACF1:         0.17
CORR:         0.81
```

Como você viu nos capítulos anteriores as métricas de avaliação de um modelo de previsão podem ser muitas e, em geral, não temos nenhuma métrica que defina este ou aquele modelo como o melhor em quaisquer casos. Mas você pode observar que os modelos de aprendizado de máquina conseguem aproximar bastante bem a série de dados do nosso exemplo quando comparado ao modelo tradicional ARIMA (veja, por exemplo o erro absoluto médio MAE) requerendo um mínimo de esforço de análise. Poderíamos ainda refinar esses modelos (empregamos aqui parâmetros padrão em todos eles) para chegar a métricas ainda melhores e podemos mesmo envolver esses modelos em laços de programa para variar os parâmetros e encontrar de modo *automático* os melhores modelos, algo que conhecemos como **AutoML** e semelhante ao que vimos no capítulo *Modelos de Previsão ARMA e ARIMA* com o pacote pmdarima de **AutoARIMA**.

Modelos Neurais

Redes neurais são regressores universais. Dada qualquer função (um mapeamento de entradas e saídas), sempre existirá uma Rede Neural que satisfaz essa função, isto é, aproxima o quanto quisermos os valores dessa função. Essa é uma capacidade muito importante dos modelos neurais e que em boa parte explica o seu sucesso em uma série aplicações modernas que vão da transcrição e tradução automática de textos no Google ou YouTube, à segmentação de imagens e cenas empregadas nos veículos autônomos da Tesla.

> **(*Kurt Hornik, Teorema da Aproximação Universal, 1991*)** Redes neurais multicamada são aproximadores universais. Dada uma função qualquer (um conjunto de treinamento), existe uma rede neural multicamada capaz de aproximar essa função o quanto se desejar.

As Redes Neurais constituem hoje uma classe bastante específica de modelos de Aprendizado de Máquina e Redes Profundas, Deep Learning, estão em geral por traz da maior parte das aplicações modernas que envolvem grandes volumes de dados e dadso multidimensionais como os exemplos da Google, YouTube e Tesla que citamos acima.

Apesar disso o modelo de um neurônio artificial é bastante simples.

Neurônio Artificial: Perceptron

Um neurônio artificial implementa uma combinação linear de entradas a qual aplica uma **função de ativação** σ para produzir uma saída.

$$f(X) = \sigma(w_0 + w_1 x_1 + \ldots + w_n x_n)$$

O treinamento do neurônio é feito ajustando-se os pesos w_n para minimizar o erro de previsão e estimar a saída $f(X) \cong y$.

$$\min_{w_i} \sum ||f(X) - y||$$

Semelhante aos coeficientes de uma regressão linear, aprendizado de um neurônio passa a ser um **problema de otimização** em buscar os valores $w_0, w_1, \ldots, w_n$ que melhor se ajustam aos dados.

Funções de Ativação

Combinações de operadores lineares são também lineares. Desse modo, sem uma função de ativação, as redes neurais seriam equivalentes a um modelo linear. As funções de ativação são uma escolha importante na arquitetura de uma rede neural e facilitam a modelagem de relações mais complexas. As funções de ativação mais comuns encontram-se abaixo.

$$f(x) = \frac{1}{(1 + e^{-x})} \quad \text{, Logistic}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad \text{, Tanh}$$

$$f(x) = max(0, x) \quad \text{, ReLu}$$

$$f(x) = log(1 + e^{x}) \quad \text{, Softplus}$$

E, não sem motivo, essa função leva o mesmo nome da função neuronal que vimos antes ;-).

Redes Neurais

Um único neurônio artificial tem uma capacidade bastante limitada representar funções. Para aumentarmos essa capacidade e podemos ter um modelo de aproximação universal esses neurônios podem ser combinados em camadas de modo que as saídas dos neurônios de uma camada possam ser entradas para camadas seguintes de neurônios.

As camadas externas são as camadas de entrada e saída da rede. As camadas internas são as camadas de ocultas (*hidden*) da rede. As redes podem ter diferentes configurações com diferentes números de entradas, saídas e número de camadas.

Essas redes são conhecidas como redes multi-camada, MLP *multi-layer perceptron* ou *feed-forward* e as entradas são processadas em sequência, uma camada após a outra.

O problema de aprendizado do modelo é, neste caso, bastante mais complexo e é obtido em um processo de iterações e aproximações sucessivas, à exemplo de um método de gradiente para o cálculo de zero de funções. O que importa é sabermos que poderão ser encontrados um conjunto de pesos $w_{j,i}$, onde j é agora a dimensão camada da rede, e que aproximarão o conjunto de dados de treinamento.

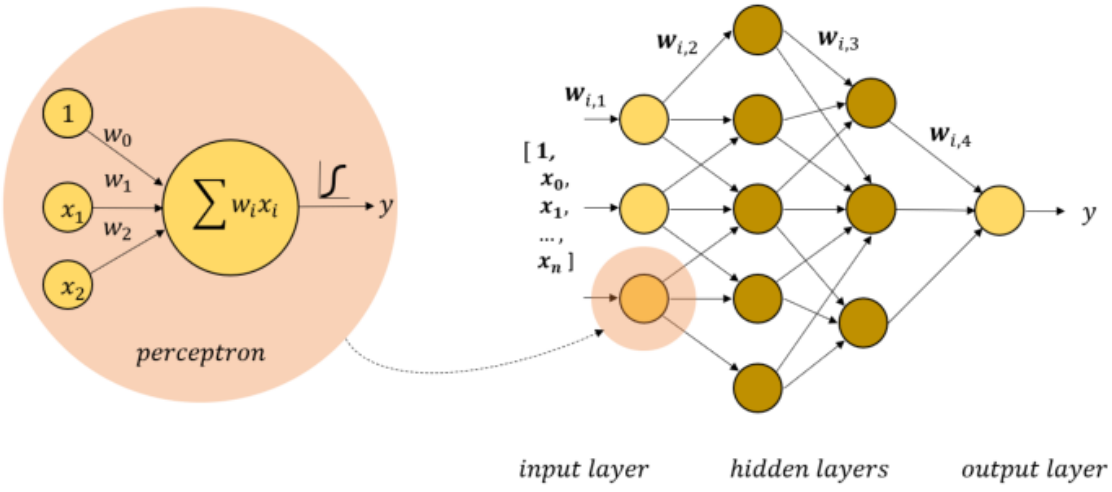


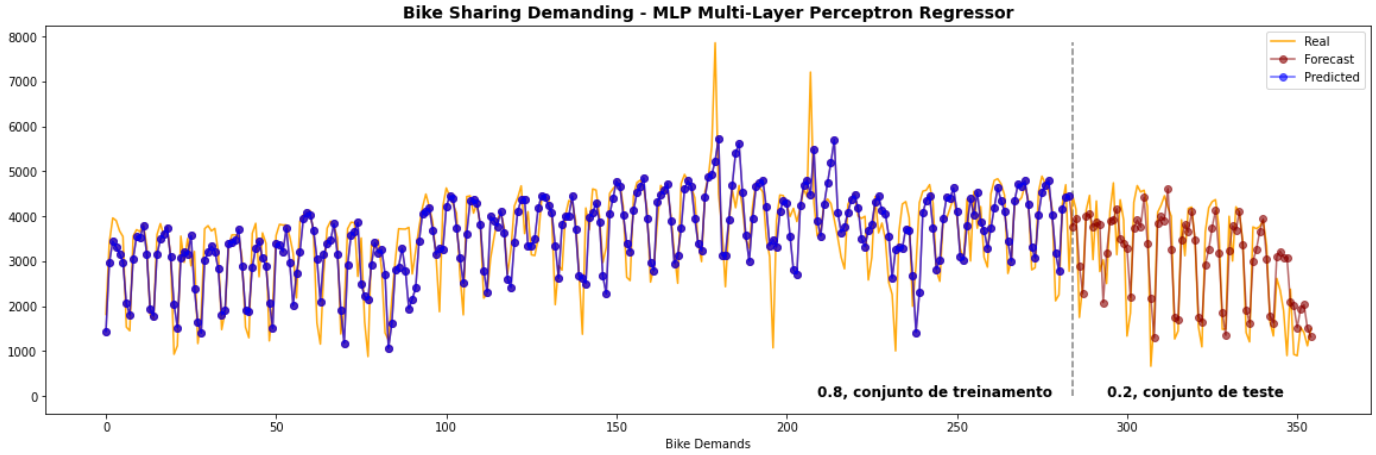
Figura 5. Um neurônio Perceptron e o Esquema de uma Rede Multicamada, MLP.

Scikit-Learn MLP

A biblioteca Scikit-Learn implementa um modelo neural multicamada MLP. A forma de aplicação é a mesma dos regressores anteriores.

```
from sklearn.neural_network import MLPRegressor

regr = MLPRegressor(hidden_layer_sizes=(8), random_state=1)
regr.fit(X_train,y_train.values.reshape(-1))

y_pred = regr.predict(X)
```

```
plot_forecast(y, y_pred, X_train, title='Bike Sharing Demanding - MLP
Multi-Layer Perceptron Regressor')

cur_metrics = forecast_accuracy(y_pred[len(y_train):],
y[len(y_train):].values.reshape(-1))
dict_metrics['MLPRegressor'] = cur_metrics
```




```
MSE:    479910.77
MAPE:   0.29
ME:     41.79
MAE:    563.62
MPE:    0.15
RMSE:   692.76
ACF1:   -0.06
CORR:   0.85

print('Mean Absolute Error')
for key, item in dict_metrics.items():
  print(f"{key} MAE: {item['MAE']:.2f}")
```

Mean Absolute Error

ARIMA Model MAE: 535.00
DecisionTreeRegressor MAE: 687.96
GradientBoostingRegressor MAE: 622.81
MLP Regressor MAE: 563.62

Do mesmo modo que nos modelos anteriores o modelo MLP aproxima bastante bem a série de dados e oderíamos refinar o modelo para obter resultados ainda melhores. Em vez disso, vamos entender como podemos empregar esses mesmos modelos para Séries Multivariadas.

Modelo MultiVariado

No capítulo anterior você estudou modelos de Séries de Dados multivariadas como o VARIMA. Nesses modelos a Série de Saída, digamos série a, depende de mais de uma Série de Dados de Entrada no tempo, digamos as séries a e b.

$$
\begin{aligned}
&\text{Lag 1:} \quad (x^a_{t-1}, x^b_{t-1}) \rightarrow x^a_t \\
&\text{Lag 2:} \quad (x^a_{t-2}, x^b_{t-2}), (x^a_{t-1}, x^b_{t-1}) \rightarrow x^a_t \\
&\text{Lag 3:} \quad (x^a_{t-3}, x^b_{t-3}), (x^a_{t-2}, x^b_{t-2}), (x^a_{t-1}, x^b_{t-1}) \rightarrow x^a_t \\
&\quad \ldots \\
&\text{Lag n:} \quad (x^a_{t-n}, x^b_{t-n}), \ldots, (x^a_{t-1}, x^b_{t-1}) \rightarrow x^a_t
\end{aligned}
$$

Os modelos de Aprendizado de Máquina reconstruir esse mapeamento de Entradas e Saídas e para isso basta adicionarmos ao nosso conjunto de dados uma nova série de entradas.

Vamos adicionar então ao nosso modelo os dados de velocidade do vento e verificar o quanto o nosso modelo melhora a sua previsão de demanda por bicicletas.

```
df_wind = ml_timeseries_df(df_resample[['wind_speed']], n_lags=7,
prefix='wind_')
df_wind.head()
```

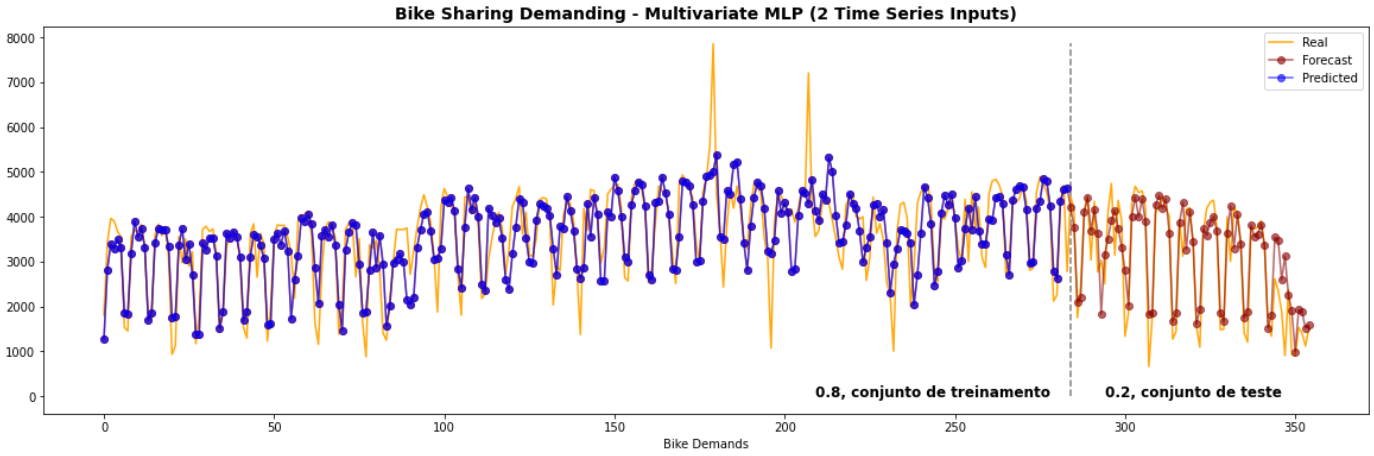
	wind_t-7	wind_t-6	wind_t-5	wind_t-4	wind_t-3	wind_t-2	wind_t-1	t
0	15.0	14.0	27.5	30.5	27.0	44.0	48.0	37.0
1	14.0	27.5	30.5	27.0	44.0	48.0	37.0	39.0
2	27.5	30.5	27.0	44.0	48.0	37.0	39.0	30.0
3	30.5	27.0	44.0	48.0	37.0	39.0	30.0	40.0
4	27.0	44.0	48.0	37.0	39.0	30.0	40.0	45.0

```
df = pd.concat([df_wind.drop(columns='t'), df],axis=1)
df.head()
```

	wind_t-7	wind_t-6	wind_t-5	wind_t-4	...	t-3	t-2	t-1	t
0	15.0	14.0	27.5	30.5	...	2351.0	3505.0	1436.0	1813
1	14.0	27.5	30.5	27.0	...	3505.0	1436.0	1813.0	3418
2	27.5	30.5	27.0	44.0	...	1436.0	1813.0	3418.0	3960
3	30.5	27.0	44.0	48.0	...	1813.0	3418.0	3960.0	3904
4	27.0	44.0	48.0	37.0	...	3418.0	3960.0	3904.0	3673

```
X = df.drop(columns='t')
y = df[['t']]
```

```
X_train, y_train, X_test, y_test = split_train_test(X, y,
test_perc=0.2)

from sklearn.neural_network import MLPRegressor

regr = MLPRegressor(hidden_layer_sizes=(8,16,8), random_state=1)
regr.fit(X_train,y_train.values.reshape(-1))

y_pred = regr.predict(X)
plot_forecast(y, y_pred, X_train, title='Bike Sharing Demanding -
Multivariate MLP (2 Time Series Inputs)')

cur_metrics = forecast_accuracy(y_pred[len(y_train):],
y[len(y_train):].values.reshape(-1))
dict_metrics['MLPRegressor with 2 input series'] = cur_metrics
```




```
MSE:    386396.82
MAPE:   0.24
ME:     126.38
MAE:    459.65
MPE:    0.16
RMSE:   621.61
ACF1:   -0.11
CORR:   0.88

print('Mean Absolute Error')
for key, item in dict_metrics.items():
  print(f"{key} MAE: {item['MAE']:.2f}")
```

Mean Absolute Error

ARIMA Model MAE: 535.00
DecisionTreeRegressor MAE: 687.96
GradientBoostingRegressor MAE: 622.81
MLPRegressor MAE: 563.62
MLP Regressor 2 input MAE: 459.65

Nosso resultado agora é até mesmo melhor que o modelo unidimensional ARIMA e você pode ver que pudemos adaptar o modelo com o mínimo de esforço.

Deep Learning

Séries Multivariadas e janelas de ciclo de tempo muito longas podem levar a Séries de Dados bastante multidimensionais. Tratar grandes volumes de dados com alta dimensionalidade exige capacidade de processamento e otimizações de código que você encontra normalmente em *Frameworks* de redes profundas como o PyTorch (Meta) ou o TensorFlow(Google), ou mesmo no pacote de Séries Temporais da Meta, o NeuralProphet.

A construção detalhada desses modelos de Deep Learning e o uso desses Frameworks estão bem além do que caberia nesta introdução, mas dada a importância e o uso cada vez maior desses modelos, cabe ao menos criarmos um modelo simples, seguindo a nossa filosofia de criar uma porta de entrada à você para modelos mais sofisticados, mas sem entrar em muitos detalhes da codificação que você poderá buscar mais adiante em outras referências.

Vamos empregar aqui o *Framework* PyTorch, sendo um dos mais empregados e bem sucedidos *Frameworks* de redes neurais profundas. Essas redes normalmente são associadas a um grande número de camadas e um grande número de elementos. Na verdade, não é exatamente assim e podemos encontrar redes Deep Learning que tenham mesmo um número pequeno de elementos e camadas (para os padrões de centenas e milhares que em geral encontramos). Uma das possibilidades que torna o uso de redes neurais profundas particularmente interessantes para a previsão de séries temporais é que podemos implementar arquiteturas recorrentes em que a saída da rede realimenta a rede, podendo-se definir ainda a profundidade dessa realimentação. Você deve ter notado a semelhança disso com os modelos de Séries Temporais.

No modelo a seguir empregaremos uma arquitetura **LSTM**, ou **Long Short-Term Memory**, que é uma arquitetura de rede neural recorrente artificia capaz de processar sequências inteiras de dados. Vamos fazer aqui uma previsão

empregando unicamente a Série de Dados de demanda de bicicletas. O código apresenta uma série de desafios técnicos para o iniciante e sugiro você seguir, em um primeiro momento, apenas os comentários do que cada parte está construindo.

Imports

```
import torch
import torch.nn as nn
from torch.autograd import Variable
```

Preparação dos Dados

Os dados serão organizados do mesmo modo que empregamos antes, embora existam, no caso das redes LSTM outras formas mais eficientes de empregarmos.

```
df = ml_timeseries_df(df_resample[['cnt']], n_lags=7)
df.head(7)
```

	t-7	t-6	t-5	t-4	t-3	t-2	t-1	t
0	1039.0	3161.0	3534.0	3618.0	2351.0	3505.0	1436.0	1813
1	3161.0	3534.0	3618.0	2351.0	3505.0	1436.0	1813.0	3418
2	3534.0	3618.0	2351.0	3505.0	1436.0	1813.0	3418.0	3960
3	3618.0	2351.0	3505.0	1436.0	1813.0	3418.0	3960.0	3904
4	2351.0	3505.0	1436.0	1813.0	3418.0	3960.0	3904.0	3673
5	3505.0	1436.0	1813.0	3418.0	3960.0	3904.0	3673.0	3546
6	1436.0	1813.0	3418.0	3960.0	3904.0	3673.0	3546.0	1531

Modelos de Deep Learning empregam os dados no formato de **Tensores** que são *arrays* multidimensionais bastante semelhantes aos empregados no NumPy mas que possuem algoritmos mais eficientes para o processamento além de acesso a recursos de GPU. Nesses modelos dos dados de entrada e saída são em geral normalizados.

```
min = df.min().min()
max = df.max().max()

for c in df:
  df[c] = ( df[c] - min ) / ( max - min )
```

Dados normalizados:

```
df.head(7)
```

	t-7	t-6	t-5	...	t-2	t-1	t
0	0.052639	0.347361	0.399167	...	0.395139	0.107778	0.160139
1	0.347361	0.399167	0.410833	...	0.107778	0.160139	0.383056
2	0.399167	0.410833	0.234861	...	0.160139	0.383056	0.458333
3	0.410833	0.234861	0.395139	...	0.383056	0.458333	0.450556

```
4  0.234861  0.395139  0.107778  ...  0.458333  0.450556  0.418472
5  0.395139  0.107778  0.160139  ...  0.450556  0.418472  0.400833
6  0.107778  0.160139  0.383056  ...  0.418472  0.400833  0.120972
```

Armazenados como Tensores e separados nos conjuntos de Treinamento e Teste como antes. Note a dimensão dos Tensores.

```
X = df.drop(columns='t')
y = df[['t']]

X = torch.tensor(np.array(X)).unsqueeze(-1)
y = torch.tensor(np.array(y))

X = X.type(torch.float)
y = y.type(torch.float)

X_train, y_train, X_test, y_test = split_train_test(X, y,
test_perc=0.2)

X.shape

torch.Size([355, 7, 1])

y.shape

torch.Size([355, 1])
```

Modelo da Rede e Treinamento

As duas funções a seguir são modelos bastante típicos de definição de uma rede e do seu treinamento e você, se tiver interesse, pode encontrar inúmeras referências e tutoriais dessas funções no site do projeto https://pytorch.org/.

Classe de Definição da Rede LSTM

```
class LSTM(nn.Module):

    def __init__(self, input_size, hidden_size, num_layers):
        super(LSTM, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
batch_first=True)
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        out, (h_out, cn) = self.lstm(x)

        h_out = h_out[-1]    # saída da última camada para a camada fc
        h_out = h_out.view(-1,self.hidden_size)
        out_final = self.fc(h_out)
```

```
        return out_final
```

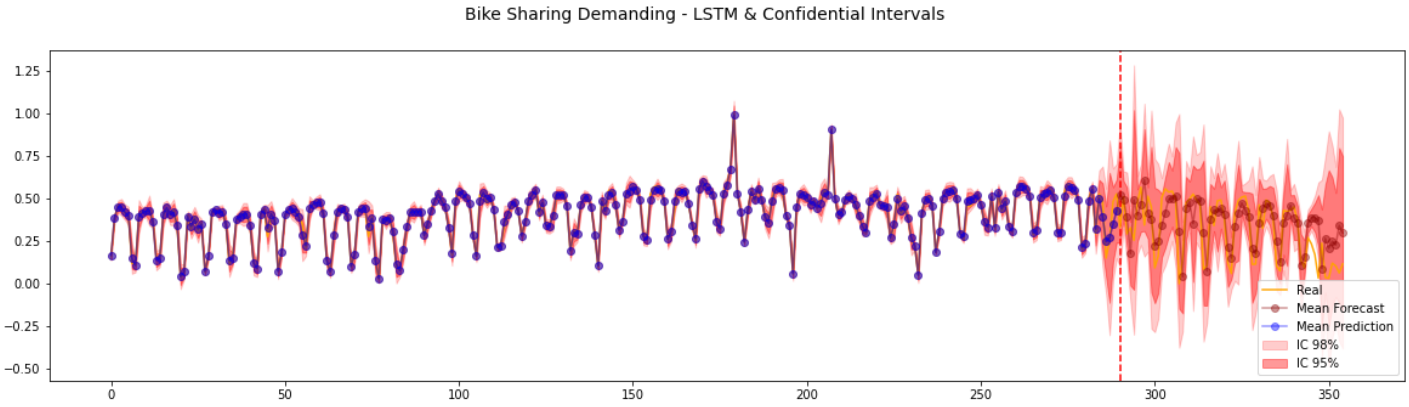
Classe de Treinamento da Rede

De modo análogo a um modelo neural em camadas o aprendizado, ou o ajuste dos pesos da rede, se dá de forma iterativa e emprega um otimizador que busca a cada etapa do treinamento encontrar valores para os pesos que minimizem o erro de saída do modelo.

```
def train(model, optmizer, loss_f, num_epochs, data_train_x,
data_train_y):

  for epoch in range(num_epochs):

    outputs = model(data_train_x)
    loss = loss_f(outputs, data_train_y)

    optimizer.zero_grad()
    loss.backward()

    optimizer.step()
    if epoch % 500 == 0:
      print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))

num_epochs = 2000
learning_rate = 0.01

input_size = 1 # input size!!! uma única série de dados
hidden_size = 8

for num_layers in range(2,3):

  model = LSTM(input_size, hidden_size, num_layers)

  loss_function = torch.nn.MSELoss()
  optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

  train(model, optimizer, loss_function, num_epochs, X_train, y_train)

  model.eval()
```

```
Epoch: 0, loss:    0.17197
Epoch: 500, loss:  0.00727
Epoch: 1000, loss: 0.00582
Epoch: 1500, loss: 0.00380
```

```
prediction = model(X)
data_predict = prediction.data.numpy()

plot_forecast(df[['t']] , data_predict, X_train, title='Bike Sharing
Demanding - LSTM Deep Learning Regressor')
```



Para obtermos as métricas nas mesmas medidas dos modelos anteriores vamos reescalar novamente os dados voltando aos valores originais antes da normalização.

```
y_forecast = data_predict[train_size::].reshape(-1) * (max - min) +
min
y_real = np.array( y[train_size::] ).reshape(-1) * (max - min) + min

cur_metrics = forecast_accuracy(y_forecast, y_real)

dict_metrics['LSTM Deep Learning Regressor'] = cur_metrics
```

```
MSE:    1284479.00
MAPE:   0.50
ME:     434.80
MAE:    848.53
MPE:    0.39
RMSE:   1133.35
ACF1:   0.18
CORR:   0.59
```

Como antes diversos ajustes poderiam ser feitos para melhorar os resultados do modelo.

Sumário do Capítulo

Aqui você conheceu e aprendeu como aplicar uma série de modelos de Aprendizado de Máquina, incluindo modelos sofisticados de Redes Neurais Profundas, na previsão de Séries Temporais. Esses modelos vem sendo cada vez mais aplicados em diversos campos e ferramentas poderosas vem empregando essas técnicas para Séries com grande volume de dados, alta dimensionalidade e não linearidade. Embora de modo prático bastante poderosos e fácil de serem aplicados (veja que independente de seu

conhecimento de Python são poucas linhas de código para implementar modelos bastante sofisticados) esses modelos requerem cuidado e, muitas vezes, não fornecem um valor explicativo sobre as Séries como vimos nos modelos mais tradicionais (AR, MA, ARMA, ARIMA etc.). Esses continuam a ser modelos essenciais de análise e recomendo que você empregue modelos de Aprendizado de Máquina em conjunto com modelos de Análise mais tradicionais. Um princípio importante da modelagem, o princípio da **Navalha de Ockham** ou **Princípio da Parcimônia**, prega que entre dois modelos que apresentem a mesma eficiência você deve optar pelo modelo mais simples, que reduz o número de hipóteses envolvidas, e uma grande parte dos problemas de Séries Temporais ainda parecem poder serem melhor resolvidos com modelos tradicionais e que fornecem ainda informações importantes da estrutura dos dados.

Apesar disso os modelos de aprendizado de máquina e redes profundas podem encontrar aplicações bastante práticas que não poderíamos resolver de outro modo e sugerimos você se familiarizar com eles.

Para Saber Mais

- O site http://playground.tensorflow.org/ permite você experimentar um modelo neural sem a necessidade de codificar. Esse *playground* está configurado inicialmente para tarefas de classificação, mas você pode setar o tipo de problema para *regressão* que é o que estudamos aqui. Lá você poderá configurar o número camadas da rede, o número de elementos, a função de ativação e acompanhar o treinamento da rede a curva de aprendizado do modelo (a evolução da função de erro).

- VANDERPLAS, Jake. **Python data science handbook: Essential tools for working with data.** O'Reilly Media, Inc., 2016. https://jakevdp.github.io/PythonDataScienceHandbook/ Acesso em: 2022.02.25.

- ZHANG, Aston et al. **Dive into deep learning.** arXiv preprint arXiv:2106.11342, 2021. https://d2l.ai/. Acesso em: 2022.02.25.

Bibliografia Comentada

As referências desta bibliografia comentada encontram-se em uma ordem que imaginamos ser a mais útil para ao leitor que busca aprofundar seus conhecimentos a partir da leitura deste texto.

Introdutórios & *Code*

- COGHLAN, A. **Welcome to a little book of R for time series.**. Parasite Genomics Group, Cambridge, U.K. https://a-little-book-of-r-for-time-series.readthedocs.io/en/latest/. Acessado em: 2022.02.25. Public Access

Este não é um livro, mas um guia prático e bastante breve do uso do R para Séries Temporais e, certamente, o ponto de entrada para quem vai iniciar a Análise de Séries Temporais com R. Embora empregue R, você pode encontrar aqui exemplos bastante simples e úteis de Séries Temporais e, ao final, e você não teria dificuldade de transpor os exemplos em R para Python.

- SHMUELI, Galit; LICHTENDAHL JR, Kenneth C. **Practical time series forecasting with r: A hands-on guide.** Axelrod schnall publishers, 2016. Access

Um livro de entrada e prático. Este é um texto que trata dos principais temas que abordados aqui de uma forma elegante e com cógidos em R concisos. Inclui capítulos de regressão lógística, redes neurais e cases. O site do livro https://www.forecastingbook.com/ traz ainda os datasets empregados, código e vídeos tutoriais.

- HYNDMAN, Rob J.; ATHANASOPOULOS, George. **Forecasting: principles and practice.** OTexts: Melbourne, Australia. OTexts.com/fpp2. Acesso em: 2022.02.25. Access | Public Access

Este é também um texto de entrada que traz a vantagem de estar editado e acessível de modo público em modo digital. Ele emprega o R, mas está concentrado no uso do pacote fable e do ggplot2 o que pode ser um desafio para iniciantes em R. Atenção especial ao capítulo Some practical forecasting issues, que discute uma série de aspectos práticos sobre predições com Séries Temporais. No site você ainda encontra materiais para intrutores, como os capítulos em .rmd e slides que podem ser bastante úteis.

- PEIXEIRO, Marco. **Time Series Forecasting in Python.** MEAP Edition. https://livebook.manning.com/book/time-series-forecasting-in-

python-book/copyright-2022-manning-publications/v-5/3. Acesso em: 2022.02.25. Access | Public Access

Um livro bastante prático que traz exemplos em Python desde os modelos tradicionais ARMA, ARIMA, ARIMA, SARIMAX até o uso de modelos de Deep Learning com o Keras-TensorFlow (Google), incluindo modelos de rede convolucionais não apresentados aqui. Os códigos de todos os capítulos também podem ser acessados no GitHub https://github.com/marcopeix/TimeSeriesForecastingInPython.

Intermediários & *Low Code*

- COWPERTWAIT, Paul SP; METCALFE, Andrew V. **Introductory time series with R.** Springer Science & Business Media, 2009. Access

Um livro mais rigoroso para quem quer se aprofundar ou explorar temas não abordados aqui. Com cógidos em R concisos o livro ainda traz capítulos de Análise Espectral, Identificação de Sistemas, Modelos Multivariados e Modelos de Espaço de Estado empregados na área de Controle e que não tratamos aqui. Os datasets empregados podem ser acessados em https://github.com/Rogerio-mack/Introductory-Time-Series-with-R-Datasets.

- NIELSEN, Aileen. **Practical time series analysis: Prediction with statistics and machine learning.** O'Reilly Media, 2019. Access

Tem um foco mais prático e voltado para o uso de técnicas de Aprendizado de Máquina, como modelos neurais, para a predição de Séries temporais. Os modelos neurais empregam o framwork MXNET (Apache). Há três capítulos de casos de aplicação em Saúde, Finanças e Governo e ainda um capítulo que discute diferentes pacotes para tratamento de Séries temporais, do Prophet (Meta) ao AnomalyDetection (Twitter). Os códigos de cada capítulo encontram-se no GitHub, https://github.com/PracticalTimeSeriesAnalysis/BookRepo.

Avançados & *No Code*

- MONTGOMERY, Douglas C.; JENNINGS, Cheryl L.; KULAHCI, Murat. **Introduction to time series analysis and forecasting**. John Wiley & Sons, 2015. Access

Apesar de um conter um apêndice de Introdução ao R, há muito poucos exemplos de código no livro. O R, entretanto, é empregado para produzir os

resultados e há ao final um sumário dos comandos em R para várias análises. São mais de 500 páginas, com destaque para o capítulo Regression Analysis and Forecasting e o que trata de funções de Transferência que são empregadas sem Identificação e Controle de Sistemas e que não foi abordado aqui. Os datasets empregados ainda podem ser obtidos aqui https://github.com/Rogerio-mack/Temporal/raw/main/Biblio/AppendixB_datafile.xls.

- MORETTIN, Pedro A.; TOLOI, Clélia. **Análise de séries temporais. In: Análise de séries temporais.** 2006. p. 538-538. Access

Esse é título que traz a vantagem de ser um dos poucos livros em língua portuguesa. Mas o título não está aqui só por isso. Ele é um texto também bastante completo de Séries Temporais e que explora, além dos modelos aqui apresentados, modelos de Espaço de Estados, Modelos não Lineares (ARCH, GARCH) empregados em análises financeiras, Análise de Fourier e Espectral.

- CHATFIELD, Chris. **The analysis of time series: an introduction.** Chapman and hall/CRC, 1996. Access

Este é um clássico cuja primeira edição é de 1989. Você não vai encontrar código nele, mas encontra um rigoroso tratamento dos temas de Séries Temporarais. Inclui ainda de capítulos de Análise Espectral, Modelos Lineares e não Lineares, e de Espaço de Estado e Filtros Kalman.

- BOX, George EP et al. **Time series analysis: forecasting and control.** John Wiley & Sons, 2015. Access

O método Box-Jenkins, como vimos aqui, se aplica aos modelos de média móvel ARMA ou ARIMA para encontrar o melhor ajuste de uma série temporal. Box e Jenkins, são dois dos autores desse texto clássico de 1996. Você pode esperar portanto encontrar um texto com um tratamento bastante rigoroso de Séries Temporais. As parte I e II do livro cobre os temas aqui apresentados enquanto as partes III e IV tratam de modelos Multivariados, Funções de Transferência e Sistemas de Controle.

- DE GOOIJER, Jan G.; HYNDMAN, Rob J. **25 years of time series forecasting.** International journal of forecasting, v. 22, n. 3, p. 443-473, 2006. Access | Public Access

Este é um artigo que revisa os modelos de séries temporarais e como eles evoluíram até bem recentemente (2006). Ele é focado nos modelos de análise tradicional e não inclui o uso de modelos neurais que surgiram mais

recentemente. É um texto curto que discute métodos de séries temporais sem entrar no detalhe de fórmulas ou suas implementações.

Outros Títulos Avançados de Interesse

- PENA, Daniel; TIAO, George C.; TSAY, Ruey S. **A course in time series analysis.** John Wiley & Sons, 2011. Access

Texto avançado, inclui ainda Análise Bayesiana de Séries Temporais, Filtros Kalman, Modelos Heteroscedásticos e modelos multivariados.

- PAOLELLA, Marc S. **Linear models and time-series analysis: regression, ANOVA, ARMA and GARCH.** John Wiley & Sons, 2018. Access

Inclui tópicos avançados como modelos GARCH, Predição de Risco e Otimização de Portfólio.

- MILLS, Terence C. **Applied time series analysis: A practical guide to modeling and forecasting.** Academic press, 2019. Access

Texto avançado que modelos de componentes não observados.

Python, Aprendizado de Máquina e Deep Learning

- KONG, Qingkai; SIAUW, Timmy; BAYEN, Alexandre. **Python Programming and Numerical Methods: A Guide for Engineers and Scientists.** Academic Press, 2020. https://pythonnumericalmethods.berkeley.edu/notebooks/Index.html. Acesso em: 2022.02.25.

- VANDERPLAS, Jake. **Python data science handbook: Essential tools for working with data.** O'Reilly Media, Inc., 2016. https://jakevdp.github.io/PythonDataScienceHandbook/ Acesso em: 2022.02.25.

- ZHANG, Aston et al. **Dive into deep learning.** arXiv preprint arXiv:2106.11342, 2021. https://d2l.ai/. Acesso em: 2022.02.25.