# prediction_quantidade

December 9, 2024

## 1 Previsão de Quantidade Diária - Versão Simplificada

Este notebook implementa um pipeline simples para prever a quantidade de vendas diária de produtos.

**Passos:** 1. Carregar e agregar dados diários. 2. Tratar datas faltantes e preencher zeros. 3. Criar features simples: dia da semana, mês, ano, feriado. 4. Dividir em treino, validação e teste. 5. Escalonar dados com base no treino. 6. Criar sequências de 30 dias para alimentar o LSTM. 7. Treinar um modelo simples LSTM para prever a quantidade do dia seguinte. 8. Avaliar no conjunto de validação e teste.

[1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import holidays
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
2024-12-09 20:32:20.950649: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1733787141.026959   39048 cuda_dnn.cc:8310] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1733787141.049549   39048 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
2024-12-09 20:32:21.222630: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

### 1.1 1. Carregar Dados

[2]:
```
csv_path = '../data/dados_transacao_26173.csv'

df = pd.read_csv(csv_path)
df['Data'] = pd.to_datetime(df['Data'], format='%Y-%m-%d')

# Agrupar por dia, somando a quantidade
daily_df = df.groupby('Data')['Quantidade'].sum().reset_index()
```

### 1.2 2. Garantir Datas Contínuas e Preencher Ausências

[3]:
```
all_dates = pd.date_range(start=daily_df['Data'].min(), end=daily_df['Data'].
  ↪max(), freq='D')
daily_df = daily_df.set_index('Data').reindex(all_dates).reset_index()
daily_df.rename(columns={'index': 'Data'}, inplace=True)

# Preencher Quantidade ausente com zero
daily_df['Quantidade'].fillna(0, inplace=True)
```

```
/tmp/ipykernel_39048/2291109107.py:6: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  daily_df['Quantidade'].fillna(0, inplace=True)
```

### 1.3 3. Feature Engineering Simples

- Vamos adicionar dia da semana, mês, ano e feriado.
- Vamos marcar feriados do Brasil.

[4]:
```
br_holidays = holidays.Brazil()

daily_df['Ano'] = daily_df['Data'].dt.year
daily_df['Mes'] = daily_df['Data'].dt.month
daily_df['DiaDaSemana'] = daily_df['Data'].dt.dayofweek  # 0 = segunda-feira
daily_df['Feriado'] = daily_df['Data'].isin(br_holidays).astype(int)

# Apenas prevemos Quantidade, aplicando log1p para estabilizar
daily_df['Quantidade_log'] = np.log1p(daily_df['Quantidade'])
```

## 1.4 4. Divisão Treino/Validação/Teste

Períodos: - Treino: 2019-01-01 até 2022-12-31 - Validação: 2023-01-01 até 2023-12-31 - Teste: 2024-01-01 até 2024-03-30

[5]:
```
train_end = pd.to_datetime('2022-12-31')
val_end = pd.to_datetime('2023-12-31')
test_start = pd.to_datetime('2024-01-01')
test_end = pd.to_datetime('2024-03-30')

train_df = daily_df[daily_df['Data'] <= train_end].copy()
val_df = daily_df[(daily_df['Data'] > train_end) & (daily_df['Data'] <=␣
 ↪val_end)].copy()
test_df = daily_df[(daily_df['Data'] >= test_start) & (daily_df['Data'] <=␣
 ↪test_end)].copy()

print("Tamanho do Treino:", len(train_df))
print("Tamanho da Validação:", len(val_df))
print("Tamanho do Teste:", len(test_df))
```

```
Tamanho do Treino: 1460
Tamanho da Validação: 365
Tamanho do Teste: 90
```

## 1.5 5. Escalonamento

Vamos escalar as features numéricas com base no conjunto de treino.
Features: `Quantidade_log`, `DiaDaSemana`, `Mes`, `Ano`, `Feriado`.

Nota: O ano pode aumentar linearmente. Podemos normalizar tudo junto para simplificar.

Vamos considerar as features da seguinte forma: Cada dia é representado por: [Quantidade_log, DiaDaSemana, Mes, Ano, Feriado]

[6]:
```
feature_cols = ['Quantidade_log', 'DiaDaSemana', 'Mes', 'Ano', 'Feriado']
# Saída: Quantidade_log do dia seguinte

scaler = MinMaxScaler()
scaler.fit(train_df[feature_cols])

train_scaled = scaler.transform(train_df[feature_cols])
val_scaled = scaler.transform(val_df[feature_cols])
test_scaled = scaler.transform(test_df[feature_cols])
```

## 1.6 6. Criação de Sequências para o LSTM

Vamos criar janelas de 30 dias para prever o dia seguinte.

Função: Para i em range(seq_length, len(data)):
X = data[i-seq_length:i, :]
y = data[i, 0] (posição 0 é Quantidade_log escalada)

[7]:
```
SEQ_LENGTH = 30

def create_sequences(data, seq_length=30):
    X, y = [], []
    # data: array de shape [dias, features]
    # A saída que queremos prever é Quantidade_log (1º feature), índice 0 no array
    for i in range(seq_length, len(data)):
        X.append(data[i-seq_length:i])  # 30 dias anteriores
        # Prever Quantidade_log do dia i (posição 0 do feature vector)
        y.append(data[i, 0])
    return np.array(X), np.array(y)

X_train, y_train = create_sequences(train_scaled, SEQ_LENGTH)
X_val, y_val = create_sequences(val_scaled, SEQ_LENGTH)
X_test, y_test = create_sequences(test_scaled, SEQ_LENGTH)

print("X_train shape:", X_train.shape, "y_train shape:", y_train.shape)
print("X_val shape:", X_val.shape, "y_val shape:", y_val.shape)
print("X_test shape:", X_test.shape, "y_test shape:", y_test.shape)
```

```
X_train shape: (1430, 30, 5) y_train shape: (1430,)
X_val shape: (335, 30, 5) y_val shape: (335,)
X_test shape: (60, 30, 5) y_test shape: (60,)
```

### 1.7 7. Construção do Modelo LSTM Simples

Uma camada LSTM pequena e uma Dense final. Sem dropout exagerado, sem camadas múltiplas inicialmente.

[8]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()
model.add(LSTM(64, input_shape=(SEQ_LENGTH, len(feature_cols))))  # sem return_sequences
model.add(Dense(1))  # Prever apenas Quantidade_log

model.compile(optimizer='adam', loss='mse')

model.summary()
```

```
I0000 00:00:1733787146.122986   39048 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 2561 MB memory:  -> device: 0,
name: NVIDIA GeForce GTX 1650, pci bus id: 0000:01:00.0, compute capability: 7.5
/home/jociano/Projects/PromoPredictor/.venv/lib/python3.12/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
```

```
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

**Model: "sequential"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| lstm (LSTM) | (None, 64) | 17,920 |
| dense (Dense) | (None, 1) | 65 |

**Total params:** 17,985 (70.25 KB)

**Trainable params:** 17,985 (70.25 KB)

**Non-trainable params:** 0 (0.00 B)

### 1.8 8. Treinamento do Modelo

Usaremos EarlyStopping para parar o treinamento se a validação não melhorar.

[9]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=20,␣
↪restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    epochs=200,
    batch_size=32,
    validation_data=(X_val, y_val),
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/200
I0000 00:00:1733787149.045014   39102 cuda_dnn.cc:529] Loaded cuDNN version
90501
45/45             3s 18ms/step -
loss: 0.2008 - val_loss: 0.0245
Epoch 2/200
45/45             0s 5ms/step - loss:
0.0225 - val_loss: 0.0162
Epoch 3/200
```

```
45/45            0s 6ms/step - loss:
0.0217 - val_loss: 0.0157
Epoch 4/200
45/45            0s 6ms/step - loss:
0.0200 - val_loss: 0.0147
Epoch 5/200
45/45            0s 5ms/step - loss:
0.0170 - val_loss: 0.0135
Epoch 6/200
45/45            0s 5ms/step - loss:
0.0235 - val_loss: 0.0138
Epoch 7/200
45/45            0s 5ms/step - loss:
0.0170 - val_loss: 0.0131
Epoch 8/200
45/45            0s 5ms/step - loss:
0.0219 - val_loss: 0.0134
Epoch 9/200
45/45            0s 5ms/step - loss:
0.0184 - val_loss: 0.0134
Epoch 10/200
45/45            0s 5ms/step - loss:
0.0183 - val_loss: 0.0130
Epoch 11/200
45/45            0s 6ms/step - loss:
0.0153 - val_loss: 0.0134
Epoch 12/200
45/45            0s 6ms/step - loss:
0.0217 - val_loss: 0.0138
Epoch 13/200
45/45            0s 5ms/step - loss:
0.0199 - val_loss: 0.0132
Epoch 14/200
45/45            0s 5ms/step - loss:
0.0165 - val_loss: 0.0127
Epoch 15/200
45/45            0s 5ms/step - loss:
0.0170 - val_loss: 0.0127
Epoch 16/200
45/45            0s 5ms/step - loss:
0.0184 - val_loss: 0.0126
Epoch 17/200
45/45            0s 5ms/step - loss:
0.0188 - val_loss: 0.0130
Epoch 18/200
45/45            0s 6ms/step - loss:
0.0206 - val_loss: 0.0126
Epoch 19/200
```

```
45/45            0s 4ms/step - loss:
0.0204 - val_loss: 0.0129
Epoch 20/200
45/45            0s 5ms/step - loss:
0.0194 - val_loss: 0.0124
Epoch 21/200
45/45            0s 5ms/step - loss:
0.0201 - val_loss: 0.0128
Epoch 22/200
45/45            0s 5ms/step - loss:
0.0197 - val_loss: 0.0127
Epoch 23/200
45/45            0s 7ms/step - loss:
0.0170 - val_loss: 0.0124
Epoch 24/200
45/45            0s 6ms/step - loss:
0.0217 - val_loss: 0.0123
Epoch 25/200
45/45            0s 5ms/step - loss:
0.0154 - val_loss: 0.0123
Epoch 26/200
45/45            0s 5ms/step - loss:
0.0191 - val_loss: 0.0123
Epoch 27/200
45/45            0s 5ms/step - loss:
0.0179 - val_loss: 0.0122
Epoch 28/200
45/45            0s 5ms/step - loss:
0.0164 - val_loss: 0.0123
Epoch 29/200
45/45            0s 5ms/step - loss:
0.0214 - val_loss: 0.0124
Epoch 30/200
45/45            0s 5ms/step - loss:
0.0239 - val_loss: 0.0128
Epoch 31/200
45/45            0s 5ms/step - loss:
0.0206 - val_loss: 0.0122
Epoch 32/200
45/45            0s 6ms/step - loss:
0.0150 - val_loss: 0.0122
Epoch 33/200
45/45            0s 5ms/step - loss:
0.0201 - val_loss: 0.0120
Epoch 34/200
45/45            0s 5ms/step - loss:
0.0195 - val_loss: 0.0124
Epoch 35/200
```

```
45/45            0s 6ms/step - loss:
0.0166 - val_loss: 0.0130
Epoch 36/200
45/45            0s 5ms/step - loss:
0.0197 - val_loss: 0.0148
Epoch 37/200
45/45            0s 5ms/step - loss:
0.0214 - val_loss: 0.0127
Epoch 38/200
45/45            0s 5ms/step - loss:
0.0179 - val_loss: 0.0151
Epoch 39/200
45/45            0s 5ms/step - loss:
0.0174 - val_loss: 0.0135
Epoch 40/200
45/45            0s 6ms/step - loss:
0.0178 - val_loss: 0.0125
Epoch 41/200
45/45            0s 5ms/step - loss:
0.0191 - val_loss: 0.0149
Epoch 42/200
45/45            0s 6ms/step - loss:
0.0172 - val_loss: 0.0122
Epoch 43/200
45/45            0s 6ms/step - loss:
0.0197 - val_loss: 0.0135
Epoch 44/200
45/45            0s 6ms/step - loss:
0.0156 - val_loss: 0.0122
Epoch 45/200
45/45            0s 7ms/step - loss:
0.0209 - val_loss: 0.0135
Epoch 46/200
45/45            0s 6ms/step - loss:
0.0159 - val_loss: 0.0144
Epoch 47/200
45/45            0s 6ms/step - loss:
0.0173 - val_loss: 0.0131
Epoch 48/200
45/45            0s 5ms/step - loss:
0.0171 - val_loss: 0.0124
Epoch 49/200
45/45            0s 5ms/step - loss:
0.0144 - val_loss: 0.0184
Epoch 50/200
45/45            0s 5ms/step - loss:
0.0208 - val_loss: 0.0129
Epoch 51/200
```

```
45/45             0s 7ms/step - loss:
0.0194 - val_loss: 0.0138
Epoch 52/200
45/45             0s 6ms/step - loss:
0.0192 - val_loss: 0.0128
Epoch 53/200
45/45             0s 5ms/step - loss:
0.0189 - val_loss: 0.0151
```

### 1.9 9. Avaliação no Conjunto de Validação

Avaliamos usando MSE e MAE. Note que a saída está em escala MinMax do log1p da quantidade. Precisamos inverter o log1p e o escalonamento para interpretar corretamente.

[10]:
```
# Vamos fazer previsões na validação
val_preds = model.predict(X_val)
# val_preds e y_val estão escalonados e log transformados

# Inverter o escalonamento e o log
# Passos:
# 1) Temos val_preds no espaço escalado. Precisamos inverter o scaler só para a␣
 ↪1ª feature (Quantidade_log).
# Para isso, criaremos um array dummy para inverter a escala.

def invert_scaling_and_log1p(scaled_values, scaler):
    # scaled_values: array shape (n_samples,) - Quantidade_log escalada
    # Precisamos recriar array com mesmas dimensões do original com features,␣
 ↪pois o scaler inverte n_features.
    temp = np.zeros((len(scaled_values), len(feature_cols)))
    temp[:, 0] = scaled_values  # colocar os valores na primeira coluna␣
 ↪(Quantidade_log)
    inv = scaler.inverse_transform(temp)
    # Agora inv[:,0] é o Quantidade_log original
    # Inverter log1p: quantidade = exp(inv[:,0]) - 1
    quant = np.expm1(inv[:,0])
    return quant

val_preds_inverted = invert_scaling_and_log1p(val_preds.flatten(), scaler)
y_val_inverted = invert_scaling_and_log1p(y_val, scaler)

mse_val = mean_squared_error(y_val_inverted, val_preds_inverted)
mae_val = mean_absolute_error(y_val_inverted, val_preds_inverted)

print("Validação - MSE:", mse_val, "MAE:", mae_val)
```

```
11/11             0s 13ms/step
Validação - MSE: 1463.0387791789865 MAE: 25.070201180139964
```

## 1.10 10. Avaliação no Conjunto de Teste

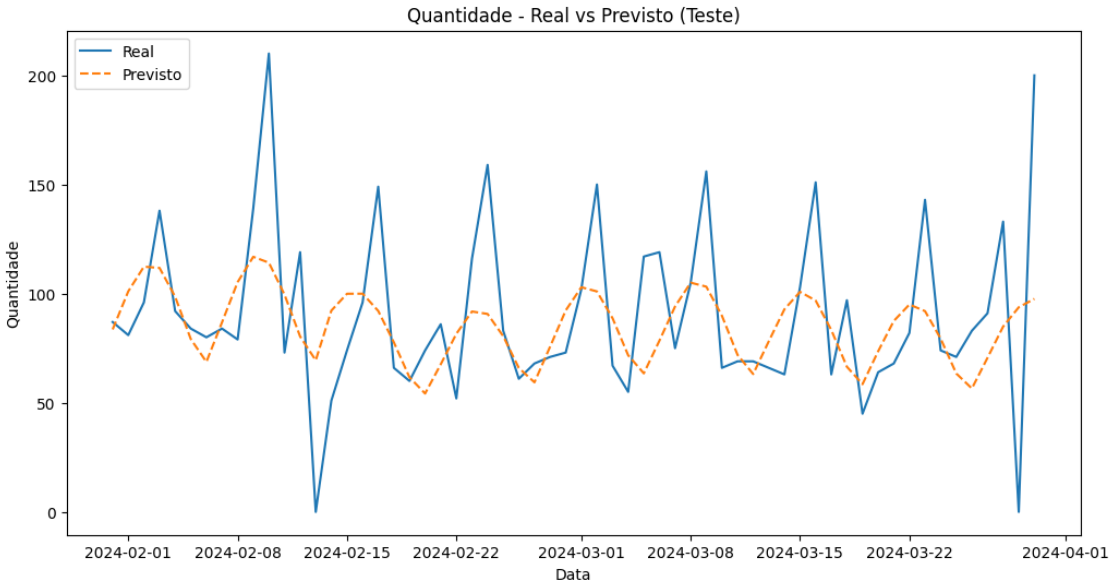
[11]:
```
test_preds = model.predict(X_test)
test_preds_inverted = invert_scaling_and_log1p(test_preds.flatten(), scaler)
y_test_inverted = invert_scaling_and_log1p(y_test, scaler)

mse_test = mean_squared_error(y_test_inverted, test_preds_inverted)
mae_test = mean_absolute_error(y_test_inverted, test_preds_inverted)

print("Teste - MSE:", mse_test, "MAE:", mae_test)
```

```
2/2             0s 2ms/step
2/2             0s 2ms/step
Teste - MSE: 1250.5542470093449 MAE: 25.983843867301054
```

## 1.11 11. Visualização das Previsões no Teste

Plotaremos a série real vs previsão para uma amostra do teste.

[12]:
```
test_dates = test_df['Data'].iloc[SEQ_LENGTH:].values  # alinhando datas com as previsões
plt.figure(figsize=(12,6))
plt.plot(test_dates, y_test_inverted, label='Real')
plt.plot(test_dates, test_preds_inverted, label='Previsto', linestyle='--')
plt.title('Quantidade - Real vs Previsto (Teste)')
plt.xlabel('Data')
plt.ylabel('Quantidade')
plt.legend()
plt.show()
```



**Conclusão**: - Temos agora um pipeline mais simples e controlado. - Caso o desempenho não seja satisfatório, podemos: - Ajustar SEQ_LENGTH - Adicionar algumas features extras (ex: encoding cíclico para mês e dia da semana). - Ajustar o tamanho do LSTM. - Uma vez que estivermos satisfeitos com a previsão de Quantidade, podemos criar outro notebook do zero apenas para ValorUnitario, seguindo um processo semelhante.