02.03-deep_learning

April 11, 2021

1 Deep Learning Model

Neste notebook tem os seguintes modelos de aprendizado de profundo comparados: - LSTM

1.1 Importações

[1]:
```
# Data analysis and data wrangling
import numpy as np
import pandas as pd

# Metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error

# Preprocessing
from sklearn.preprocessing import MinMaxScaler

# Plotting
import seaborn as sns
import matplotlib.pyplot as plt

# deep learning
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

# Other
from IPython.display import Image
import warnings
import pprint
import datetime
import os
import datetime
```

1.2 Preparação do Diretório Principal

[2]:
```python
def prepare_directory_work(end_directory: str='notebooks'):
    # Current path
    curr_dir = os.path.dirname (os.path.realpath ("__file__"))

    if curr_dir.endswith(end_directory):
        os.chdir('..')
        return curr_dir

    return f'Current working directory: {curr_dir}'
```

[3]:
```python
prepare_directory_work(end_directory='notebooks')
```

[3]: '/home/campos/projects/tcc-ufsc-grad/notebooks'

1.3 Formatação das Células

[4]:
```python
# OPTIONAL: Load the "autoreload" extension so that code can change
%load_ext autoreload

# Guarantees visualization inside the jupyter
%matplotlib inline

# Print xxxx rows and columns
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('float_format', '{:f}'.format)

# Supress unnecessary warnings so that presentation looks clean
warnings.filterwarnings('ignore')

# pretty print
pp = pprint.PrettyPrinter(indent=4)
```

[5]:
```python
plt.style.use('seaborn')  # fivethirtyeight
plt.rc('figure',figsize=(16,8))
plt.rc('font',size=15)
plt.rc('legend',fontsize=15)

# Seaborn rcParams
# ===================
sns.set(context='poster',  # notebook
        style='darkgrid',
        palette='deep',
        color_codes=True)
```

```python
# graph style
sns.set(style='dark', palette='deep')

plt.style.use('fivethirtyeight')
```

1.4 Carregamento dos Dados

[6]:
```python
%%time

df_vale3 = pd.read_csv('data/cleansing/df_vale3_cleansing.csv',
                       encoding='utf8',
                       delimiter=',',
                       parse_dates=True,
                       index_col=0,
                       verbose=True)
```

```
Tokenization took: 1.69 ms
Type conversion took: 2.27 ms
Parser memory cleanup took: 0.00 ms
CPU times: user 8.33 ms, sys: 5.54 ms, total: 13.9 ms
Wall time: 12.6 ms
```

[7]:
```python
print(df_vale3.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2445 entries, 2010-07-12 to 2020-05-28
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   preco         2445 non-null   float64
 1   residuos      2445 non-null   float64
 2   tendencia     2445 non-null   float64
 3   sazonalidade  2445 non-null   float64
 4   diff_1        2445 non-null   float64
 5   diff_2        2445 non-null   float64
 6   diff_3        2445 non-null   float64
 7   diff_4        2445 non-null   float64
 8   diff_5        2445 non-null   float64
dtypes: float64(9)
memory usage: 191.0 KB
None
```

[8]:
```python
df_vale3.head()
```

[8]:
```
                preco  residuos  tendencia  sazonalidade    diff_1    diff_2  \
data
2010-07-12  40.000000  1.002310  41.827333      1.000149 -0.600000 -0.460000
2010-07-13  40.070000  1.036654  41.910833      0.998563  0.070000 -0.530000
2010-07-14  40.080000  1.028377  41.977833      1.000439  0.010000  0.080000
2010-07-15  39.760000  1.044658  42.045833      1.000935 -0.320000 -0.310000
2010-07-16  38.880000  1.028132  42.123500      1.001784 -0.880000 -1.200000

               diff_3    diff_4    diff_5
data
2010-07-12   0.490000  0.980000  0.420000
2010-07-13  -0.390000  0.560000  1.050000
2010-07-14  -0.520000 -0.380000  0.570000
2010-07-15  -0.240000 -0.840000 -0.700000
2010-07-16  -1.190000 -1.120000 -1.720000
```

1.5 Divisão dos Dados

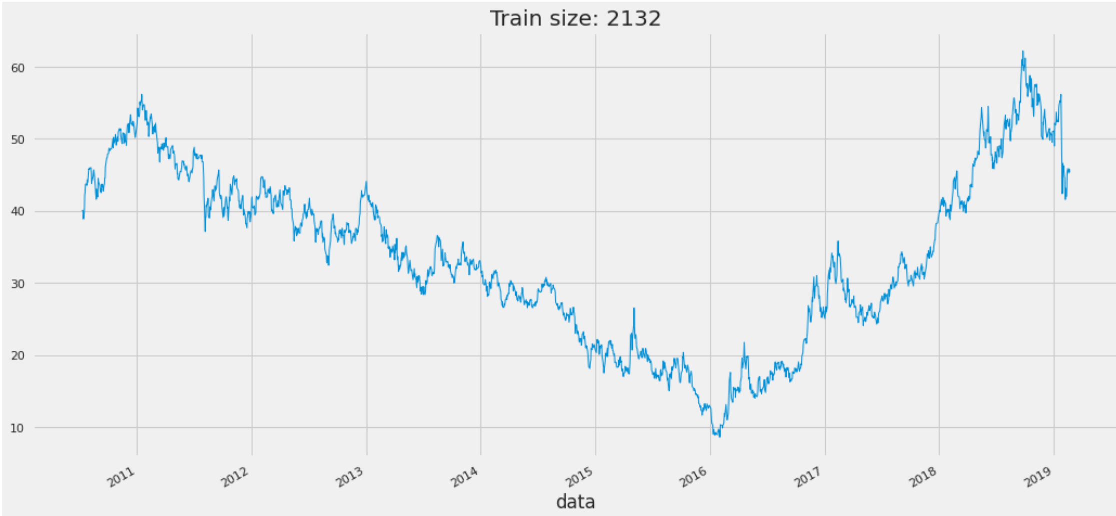
[9]:
```python
size_train = 2132
size_test = 313
print(size_train)
print(size_test)

df_train = df_vale3.iloc[:size_train]
df_test = df_vale3.iloc[size_train:]
print(df_train.columns)
print(df_test.columns)
```

```
2132
313
Index(['preco', 'residuos', 'tendencia', 'sazonalidade', 'diff_1', 'diff_2',
       'diff_3', 'diff_4', 'diff_5'],
      dtype='object')
Index(['preco', 'residuos', 'tendencia', 'sazonalidade', 'diff_1', 'diff_2',
       'diff_3', 'diff_4', 'diff_5'],
      dtype='object')
```

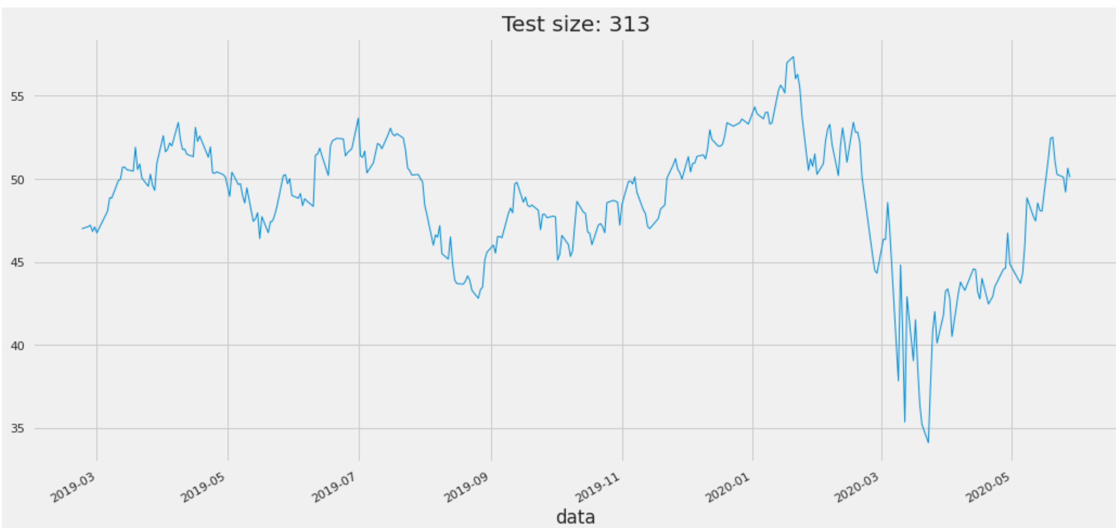
[10]:
```python
df_train['preco'].plot(linewidth=1)
plt.grid(True)
plt.title(f'Train size: {len(df_train)}')
```

[10]:
```
Text(0.5, 1.0, 'Train size: 2132')
```



[11]:
```
df_test['preco'].plot(linewidth=1)
plt.grid(True)
plt.title(f'Test size: {len(df_test)}')
```

[11]: Text(0.5, 1.0, 'Test size: 313')



[12]:
```
df_train.index
```

[12]:
```
DatetimeIndex(['2010-07-12', '2010-07-13', '2010-07-14', '2010-07-15',
               '2010-07-16', '2010-07-19', '2010-07-20', '2010-07-21',
               '2010-07-22', '2010-07-23',
               …
```

```
               '2019-02-08', '2019-02-11', '2019-02-12', '2019-02-13',
               '2019-02-14', '2019-02-15', '2019-02-18', '2019-02-19',
               '2019-02-20', '2019-02-21'],
              dtype='datetime64[ns]', name='data', length=2132, freq=None)
```

[13]:
```
df_test.index
```

[13]:
```
DatetimeIndex(['2019-02-22', '2019-02-25', '2019-02-26', '2019-02-27',
               '2019-02-28', '2019-03-01', '2019-03-06', '2019-03-07',
               '2019-03-08', '2019-03-11',
               …
               '2020-05-15', '2020-05-18', '2020-05-19', '2020-05-20',
               '2020-05-21', '2020-05-22', '2020-05-25', '2020-05-26',
               '2020-05-27', '2020-05-28'],
              dtype='datetime64[ns]', name='data', length=313, freq=None)
```

1.6 Dicionário de Resultados

[14]:
```
dict_results = {}
```

1.7 Impressão dos Resutados

[15]:
```
def show_result_model(df_train, df_test, y_forecast, model_name):
    future_forecast = pd.DataFrame(y_forecast,
                                   index=df_test.index,
                                   columns=['previsao'])
    #mape = mean_absolute_percentage_error(df_test, y_forecast)
    mape = mean_absolute_percentage_error(df_test, y_forecast)*100

    mse = mean_squared_error(df_test, y_forecast, squared=True)
    dict_results[model_name] = [mape, mse]

    pd.concat([df_test, future_forecast], axis=1).plot()

    plt.legend()
    plt.grid(True)
    plt.xlabel("Tempo (dias)", fontsize=20)
    plt.ylabel("Preço (R$)", fontsize=20)
    plt.title(f'MAPE = {mape:.2f} % | MSE = {mse:.2f}', fontsize=25)
```

1.8 Normalização dos Dados

[16]:
```
train_max = df_train.max()
train_min = df_train.min()

train = (df_train - train_min)/(train_max - train_min)
test = (df_test - train_min)/(train_max - train_min)
```

[17]:
```
def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []

    for i in range(len(X) - time_steps):
        v = X.iloc[i:(i + time_steps)].values
        Xs.append(v)
        ys.append(y.iloc[i + time_steps])

    return np.array(Xs).astype('float32'), np.array(ys).astype('float32')
```

[18]:
```
time_steps = 1

X_train, y_train = create_dataset(train, train['preco'], time_steps)
X_test, y_test = create_dataset(test, test['preco'], time_steps)
```

1.9 LSTM

- reference: https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

[19]:
```
# sequential model
model_lstm = Sequential(name='lstm_vale3')
model_lstm
```

[19]:
```
<tensorflow.python.keras.engine.sequential.Sequential at 0x7f32dbbc8a00>
```

Input Layer

[20]:
```
#Adding the first LSTM layer and some Dropout regularisation
model_lstm.add(LSTM(units=len(df_train.columns),
                    return_sequences=True,
                    input_shape=(X_train.shape[1], X_train.shape[2])))
model_lstm.add(Dropout(0.2))
```

Hidden Layers

[21]:
```
# Adding a second LSTM layer and some Dropout regularisation
model_lstm.add(LSTM(units=10, return_sequences=True))
model_lstm.add(Dropout(0.2))
```

```
# Adding a third LSTM layer and some Dropout regularisation
model_lstm.add(LSTM(units=10, return_sequences=True))
model_lstm.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation
model_lstm.add(LSTM(units=10))
model_lstm.add(Dropout(0.2))
```

Output Layer

[22]:
```
model_lstm.add(Dense(units=1))
```

1.9.1 Compilação da RNA

[23]:
```
model_lstm.compile(loss='mean_squared_error',
                   optimizer='adam',
                   metrics=['mse', 'mape'])
```

1.9.2 Resumo da RNA

[24]:
```
model_lstm.summary()
```

```
Model: "lstm_vale3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 1, 9)              684
_________________________________________________________________
dropout (Dropout)            (None, 1, 9)              0
_________________________________________________________________
lstm_1 (LSTM)                (None, 1, 10)             800
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 10)             0
_________________________________________________________________
lstm_2 (LSTM)                (None, 1, 10)             840
_________________________________________________________________
dropout_2 (Dropout)          (None, 1, 10)             0
_________________________________________________________________
lstm_3 (LSTM)                (None, 10)                840
_________________________________________________________________
dropout_3 (Dropout)          (None, 10)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 11
=================================================================
Total params: 3,175
Trainable params: 3,175
```

```
Non-trainable params: 0
_________________________________________________________________
```

1.10 Treinamento

- batch_size: cria lote de treinamento de 30 em 30 dias

[25]:
```
%%time


history = model_lstm.fit(X_train,
               y_train,
               epochs=1000,
               batch_size=30,
               shuffle=False,
               validation_split=0.30,
               verbose=0)
history
```

```
CPU times: user 9min 51s, sys: 1min 14s, total: 11min 5s
Wall time: 5min 3s
```

[25]:
```
<tensorflow.python.keras.callbacks.History at 0x7f32dbc2bb50>
```

[26]:
```
print(history.history.keys())
```

```
dict_keys(['loss', 'mse', 'mape', 'val_loss', 'val_mse', 'val_mape'])
```

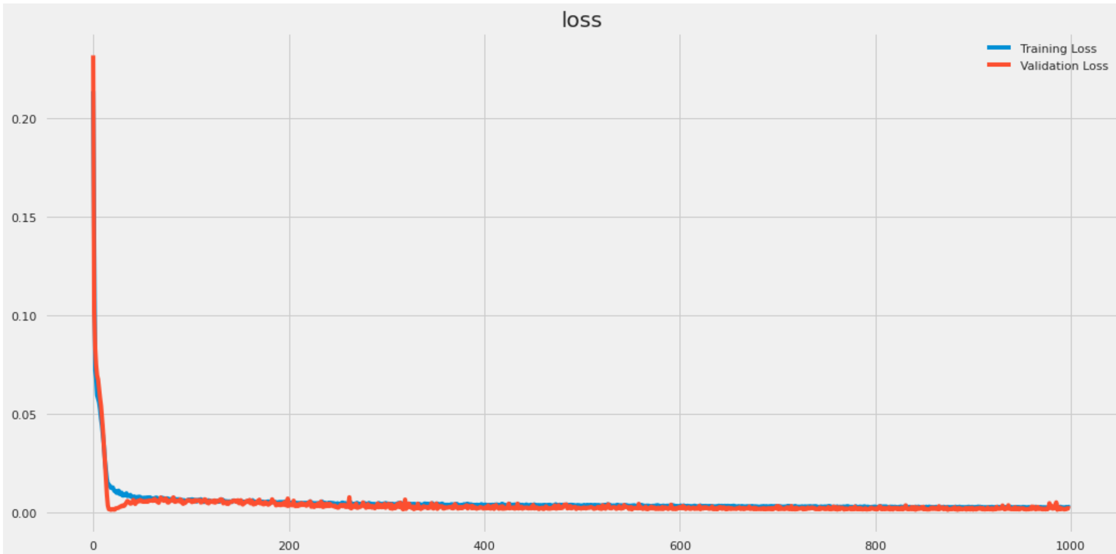
[27]:
```
best_epochs = history.history["loss"].index(min(history.history["loss"]))
best_epochs
```

[27]:
```
996
```

[28]:
```
min(history.history["loss"])
```

[28]:
```
0.002315280493348837
```

[29]:
```
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.title('loss')
plt.legend()
plt.show()
```



1.10.1 Previsão

[30]:
```
y_pred = model_lstm.predict(X_test)
```

[31]:
```
# Rescale the data back to the original scale
y_test = y_test*(train_max[0] - train_min[0]) + train_min[0]
y_pred = y_pred*(train_max[0] - train_min[0]) + train_min[0]
y_train = y_train*(train_max[0] - train_min[0]) + train_min[0]
```

[32]:
```
y_test[:10]
```

[32]:
```
array([47.120003, 47.199997, 46.83    , 47.1     , 46.739998, 48.049995,
       48.86    , 48.85    , 49.879997, 49.97    ], dtype=float32)
```

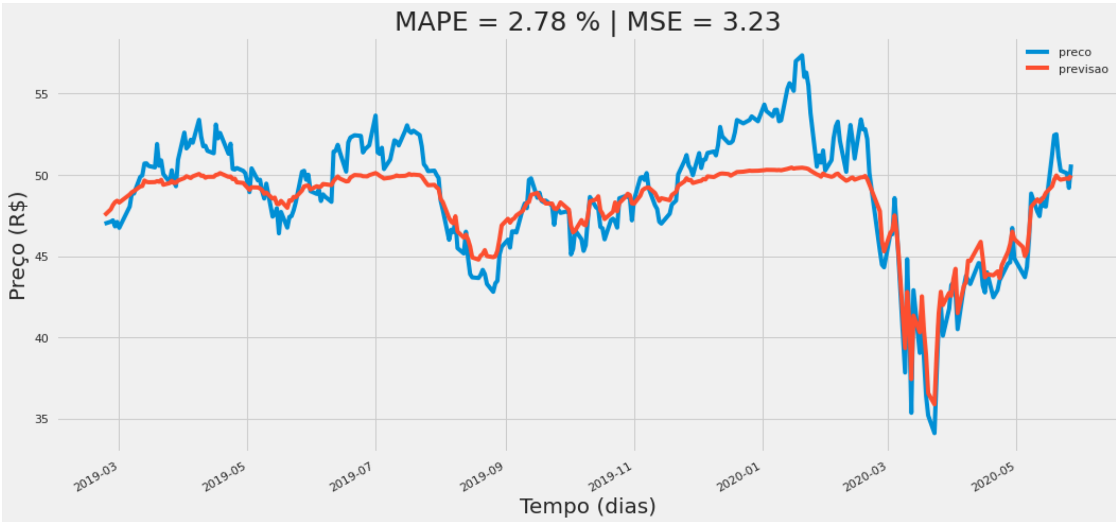
[33]:
```
len(y_test)
```

[33]:
```
312
```

[34]:
```
y_train[:10]
```

[34]:
```
array([40.07    , 40.079998, 39.760002, 38.879997, 39.969997, 42.229996,
       42.47    , 43.370003, 43.71    , 43.730003], dtype=float32)
```

[37]:
```
show_result_model(df_train=df_test['preco'][:312],
                  df_test=df_test['preco'][:312],
                  y_forecast=y_pred[:312],
                  model_name='model_lstm')
```



1.11 Resultados

[38]:
```
dict_results
```

[38]:
```
{'model_lstm': [2.7826121385151543, 3.2280584683956315]}
```
