Trabalho Final de Previsão de Ativos Financeiros

Análise Preditiva Avançada

MBA Executivo em Business Analytics e Big Data - Campinas

FGV

Integrantes do Grupo:

Adriana Marques Guidi, matrícula: A58318039

Érico Daniel Witzel dos Reis, matrícula: A58239101

Mário Elias Marinho Vieira, matrícula: A57860064

Rodrigo Kenji Makita Nakamura, matrícula: A57891680

O objetivo deste trabalho é criar um modelo para prever se, no próximo minuto, o mini indice (WIN) irá subir 10 pontos ou mais. O dado fornecido é a série histórica, minuto a minuto, do preço do ativo e o volume negociado.

Como escopo deste trabalho, serão desenvolvidos quatro modelos para essa predição, sendo eles: Naive, Naive assumindo sempre zero, Redes Neurais Artificias e XGBOOST (tipo árvore). Esse código final, apresenta o Naive, Naive zero e XGBOOST como "código" e a rede neural é apresentada como Markdown nessa última versão!

In [11]:

```
####################################
### Importação de pacotes PYTHON ###
####################################
# geral
import pandas as pd
import numpy as np
from ta.utils import dropna
from ta.volatility import BollingerBands
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
# rede neural
from tensorflow import keras
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.optimizers import SGD
%matplotlib inline

########################################
### Importação da base do mini-índice ###
########################################
data = pd.read_csv('data.csv', sep='\t')

#############################################################################
### Definindo funções básicas - acuracia e separação de treino e teste ###
#############################################################################
# função acuracia
def acuracia(y_real, y_pred):
    return sum(y_real == y_pred) / len(y_pred)

# função para Separação de treino e teste
def split_treino_test(data):
    train = data[data['datetime'] < '2020-07-01']
    teste = data[data['datetime'] >= '2020-07-01']
    return train, teste

####################################
### Tratamentos básicos de datas ###
####################################
data['datetime'] = data['DATE'] + ' ' + data['TIME']
data['datetime'] = pd.to_datetime(data['datetime'])
# Ordena base de dados por datetime
data.sort_values(by = 'datetime', inplace = True)
# Mantém apenas as variáveis datetime, tickvol e close
data = data[['datetime', 'TICKVOL', 'CLOSE']]

########################################
### Criando variável resposta target ###
########################################
data.rename(columns = {'CLOSE':'valor'}, inplace = True)
data.loc[:,'target_num'] = data['valor'].shift(-1)
data.loc[:,'target_diff'] = data['target_num']  - data['valor']
# cria variável target pela diferença entre a leitura do close atual
# com a próxima leitura (leitura de 1 minuto posterior)
data.loc[:,'target'] = data['target_diff'].apply(lambda x: 1 if x>=10 else 0)
data = data.drop(columns = ['target_num', 'target_diff'])

########################################
### remoção da primeira linha do dia ###
########################################
data['primeira_do_dia'] = data['datetime'].diff() > pd.Timedelta(10, unit = 'hour')
```

```
data = data.loc[data['primeira_do_dia'] == False]
data = data[['datetime', 'valor', 'TICKVOL','target']].reset_index(drop = True)

###################################
### Criação de modelos baseline ###
###################################

#Criação de um modelo baseline utilizando 'naive', para servir como comparação para os mode
# Criação de variável target com lag
_, teste = split_treino_test(data)

teste_naive = teste.copy()

teste_naive['target_l1'] = teste_naive['target'].shift(1)
teste_naive=teste_naive.dropna()

# A predição é simplesmente o time step anterior
performance_baseline = acuracia(teste_naive['target'], teste_naive['target_l1'])
print("A acurácia do modelo Naive é " + str(round(100*performance_baseline,1)) + "%")

# Outra predição Naive seria assumir que todos os resultados são 0:
qtdade_zeros_base = 1-teste_naive['target'].mean()
print("A acurácia assumindo que todos são 0 é " + str(round(100*qtdade_zeros_base,1)) + "%"
```

```
A acurácia do modelo Naive é 52.5%
A acurácia assumindo que todos são 0 é 57.7%
```

In [2]:

```
pip install -U numpy
```

```
Requirement already satisfied: numpy in /srv/conda/envs/notebook/lib/python
3.7/site-packages (1.18.5)
Collecting numpy
  Downloading numpy-1.21.1-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x8
6_64.whl (15.7 MB)
     |████████████████████████████████| 15.7 MB 4.2 MB/s eta 0:00:01
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.18.5
    Uninstalling numpy-1.18.5:
      Successfully uninstalled numpy-1.18.5
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
tensorflow 2.3.0 requires numpy<1.19.0,>=1.16.0, but you have numpy 1.21.1 w
hich is incompatible.
Successfully installed numpy-1.21.1
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
######################################################################
### Criação de variáveis baseadas nos fechamentos e no tickvol ###
######################################################################

########################################
### criando uma cópia da base inicial ###
########################################
data_1 = data.copy()

###################################
### variáveis baseadas em close ###
###################################
d = pd.Series(data["valor"])

# variáveis de médias móveis 20 e 10 minutos
data_1['c_ma20'] = d.rolling(20).mean()
data_1['c_ma10'] = d.rolling(10).mean()

# comparando média móvel de 20 minutos com média móvel de 10 minutos
data_1['c_ma10/ma20'] = data_1['c_ma10']/data_1['c_ma20']
data_1['c_ma20_maior_ma10'] = np.where((data_1.c_ma20 > data_1.c_ma10),1,0)

# variáveis com máximo e mínimo de 20 e 10 minutos
data_1['c_max20'] = d.rolling(20).max()
data_1['c_max10'] = d.rolling(10).max()
data_1['c_min20'] = d.rolling(20).min()
data_1['c_min10'] = d.rolling(10).min()
data_1['c_max10/max20'] = data_1['c_max10']/data_1['c_max20']
data_1['c_max10/min20'] = data_1['c_max10']/data_1['c_min20']
data_1['c_min10/max20'] = data_1['c_min10']/data_1['c_max20']
data_1['c_min10/min20'] = data_1['c_min10']/data_1['c_min20']
data_1['c_min10/max10'] = data_1['c_min10']/data_1['c_max10']
data_1['c_min20/max20'] = data_1['c_min20']/data_1['c_max20']

# comparações entre mínimo, máximo e médias móveis
data_1['c_ma10/max20'] = data_1['c_ma10']/data_1['c_max20']
data_1['c_ma10/ma20'] = data_1['c_ma10']/data_1['c_ma20']
data_1['c_ma10/min20'] = data_1['c_ma10']/data_1['c_min20']
data_1['c_ma10/min10'] = data_1['c_ma10']/data_1['c_min10']
data_1['c_ma10/max10'] = data_1['c_ma10']/data_1['c_max10']
data_1['c_ma20/max20'] = data_1['c_ma20']/data_1['c_max20']
data_1['c_ma20/min20'] = data_1['c_ma20']/data_1['c_min20']
data_1['c_ma20/min10'] = data_1['c_ma20']/data_1['c_min10']
data_1['c_ma20/max10'] = data_1['c_ma20']/data_1['c_max10']
data_1['c_ma10_maior_max20'] = np.where((data_1.c_ma10 > data_1.c_max20),1,0)
data_1['c_ma20_menor_min10'] = np.where((data_1.c_ma20 < data_1.c_min10),1,0)
data_1['c_ma10_menor_min20'] = np.where((data_1.c_ma10 < data_1.c_min20),1,0)

# comparações entre ponto atual e máximos e mínimos de 10 e 20 minutos
data_1['c_valor_menor_min20'] = np.where((data_1.valor < data_1.c_min20),1,0)
data_1['c_valor_menor_min10'] = np.where((data_1.valor < data_1.c_min10),1,0)
data_1['c_valor_maior_max20'] = np.where((data_1.valor > data_1.c_max20),1,0)
data_1['c_valor_maior_max10'] = np.where((data_1.valor > data_1.c_max10),1,0)

# criação das bollinger bands considerando n = 20 minutos e 2 desvios padrão
indicator_bb = BollingerBands(close=data_1["valor"], n=20, ndev=2)
data_1['c_bb_bbp'] = indicator_bb.bollinger_pband()
data_1['c_bb_upper'] = indicator_bb.bollinger_hband_indicator()
data_1['c_bb_lower'] = indicator_bb.bollinger_lband_indicator()
```

```
# criação do fechamento lagado de até 3 minutos
data_1['c_valor_l1'] = data_1['valor'].shift(1)
data_1['c_valor_l2'] = data_1['valor'].shift(2)
data_1['c_valor_l3'] = data_1['valor'].shift(3)
# comparação entre o valor de fechamento atual e os fechamentos lagados
data_1['c_diff_l1'] = data_1['valor']  - data_1['c_valor_l1']
data_1['c_diff_l2'] = data_1['valor']  - data_1['c_valor_l2']
data_1['c_diff_l3'] = data_1['valor']  - data_1['c_valor_l3']
# comparação entre o valor de fechamento lagado de 1 minuto e fechamento lagado de 20 minut
data_1['c_lag1/lag20'] = data_1['valor'].shift(1)/data_1['valor'].shift(20)
data_1['c_lag1/lag10'] = data_1['valor'].shift(1)/data_1['valor'].shift(10)
data_1['c_coef_ang20'] = (data_1['c_valor_l1'] - data_1['valor'].shift(20))/20
data_1['c_coef_ang10'] = (data_1['c_valor_l1'] - data_1['valor'].shift(10))/10

# quantidade de vezes que o índice fechou positivo ou negativo em relação ao minuto anterio
# nos últimos 20 minutos e 10 minutos
x = 0
y = 0
for i in range(1,19):
    x = np.where((data_1['valor'].shift(i) > data_1['valor'].shift(i+1)),1,0) + x
    y = np.where((data_1['valor'].shift(i) < data_1['valor'].shift(i+1)),1,0) + y

a = 0
b = 0
for i in range(1,9):
    a = np.where((data_1['valor'].shift(i) > data_1['valor'].shift(i+1)),1,0) + a
    b = np.where((data_1['valor'].shift(i) < data_1['valor'].shift(i+1)),1,0) + b

data_1['c_qtde_vezes_positivo20'] = x
data_1['c_qtde_vezes_negativo20'] = y
data_1['c_qtde_vezes_positivo10'] = a
data_1['c_qtde_vezes_negativo10'] = b

# variáveis lagadas vs mínimos, média móvel
data_1['c_lag1/min20'] = data_1['valor'].shift(1)/data_1['c_min20']
data_1['c_lag2/min20'] = data_1['valor'].shift(2)/data_1['c_min20']
data_1['c_lag3/min20'] = data_1['valor'].shift(3)/data_1['c_min20']
data_1['c_lag1/ma20'] = data_1['valor'].shift(1)/data_1['c_ma20']

# variável média vs máximo
data_1['c_ma3/max3'] = d.rolling(3).mean()/d.rolling(3).max()

# normalizando
media = data_1['c_ma20'].mean()
data_1['c_ma20'] = data_1['c_ma20']/media
media = data_1['c_ma10'].mean()
data_1['c_ma10'] = data_1['c_ma10']/media

media = data_1['c_max20'].mean()
data_1['c_max20'] = data_1['c_max20']/media
media = data_1['c_max10'].mean()
data_1['c_max10'] = data_1['c_max10']/media
media = data_1['c_min20'].mean()
data_1['c_min20'] = data_1['c_min20']/media
media = data_1['c_min10'].mean()
data_1['c_min10'] = data_1['c_min10']/media

media = data_1['c_bb_upper'].mean()
data_1['c_bb_upper'] = data_1['c_bb_upper']/media
media = data_1['c_bb_lower'].mean()
```

```
data_1['c_bb_lower'] = data_1['c_bb_lower']/media

media = data_1['c_valor_l1'].mean()
data_1['c_valor_l1'] = data_1['c_valor_l1']/media
media = data_1['c_valor_l2'].mean()
data_1['c_valor_l2'] = data_1['c_valor_l2']/media
media = data_1['c_valor_l3'].mean()
data_1['c_valor_l3'] = data_1['c_valor_l3']/media

media = data_1['c_diff_l1'].mean()
data_1['c_diff_l1'] = data_1['c_diff_l1']/media
media = data_1['c_diff_l2'].mean()
data_1['c_diff_l2'] = data_1['c_diff_l2']/media
media = data_1['c_diff_l3'].mean()
data_1['c_diff_l3'] = data_1['c_diff_l3']/media

media = data_1['c_coef_ang20'].mean()
data_1['c_coef_ang20'] = data_1['c_coef_ang20']/media
media = data_1['c_coef_ang10'].mean()
data_1['c_coef_ang10'] = data_1['c_coef_ang10']/media

media = data_1['c_qtde_vezes_positivo20'].mean()
data_1['c_qtde_vezes_positivo20'] = data_1['c_qtde_vezes_positivo20']/media
media = data_1['c_qtde_vezes_negativo20'].mean()
data_1['c_qtde_vezes_negativo20'] = data_1['c_qtde_vezes_negativo20']/media
media = data_1['c_qtde_vezes_positivo10'].mean()
data_1['c_qtde_vezes_positivo10'] = data_1['c_qtde_vezes_positivo10']/media
media = data_1['c_qtde_vezes_negativo10'].mean()
data_1['c_qtde_vezes_negativo10'] = data_1['c_qtde_vezes_negativo10']/media

media = data_1['valor'].mean()
data_1['valor'] = data_1['valor']/media

#####################################
### variáveis baseadas em tickvol ###
#####################################
a = pd.Series(data["TICKVOL"])

# variáveis de médias móveis 20 e 10 minutos
data_1['t_ma20'] = a.rolling(20).mean()
data_1['t_ma10'] = a.rolling(10).mean()
# comparando média móvel de 20 minutos com média móvel de 10 minutos
data_1['t_ma10/ma20'] = data_1['t_ma10']/data_1['t_ma20']
data_1['t_ma20_maior_ma10'] = np.where((data_1.t_ma20 > data_1.t_ma10),1,0)

# variáveis com máximo e mínimo de 20 e 10 minutos
data_1['t_max20'] = a.rolling(20).max()
data_1['t_max10'] = a.rolling(10).max()
data_1['t_min20'] = a.rolling(20).min()
data_1['t_min10'] = a.rolling(10).min()
data_1['t_max10/max20'] = data_1['t_max10']/data_1['t_max20']
data_1['t_max10/min20'] = data_1['t_max10']/data_1['t_min20']
data_1['t_min10/max20'] = data_1['t_min10']/data_1['t_max20']
data_1['t_min10/min20'] = data_1['t_min10']/data_1['t_min20']
data_1['t_min10/max10'] = data_1['t_min10']/data_1['t_max10']
data_1['t_min20/max20'] = data_1['t_min20']/data_1['t_max20']

# comparações entre mínimo, máximo e médias móveis
data_1['t_ma10/max20'] = data_1['t_ma10']/data_1['t_max20']
data_1['t_ma10/ma20'] = data_1['t_ma10']/data_1['t_ma20']
data_1['t_ma10/min20'] = data_1['t_ma10']/data_1['t_min20']
```

```
data_1['t_ma10/min10'] = data_1['t_ma10']/data_1['t_min10']
data_1['t_ma10/max10'] = data_1['t_ma10']/data_1['t_max10']
data_1['t_ma20/max20'] = data_1['t_ma20']/data_1['t_max20']
data_1['t_ma20/min20'] = data_1['t_ma20']/data_1['t_min20']
data_1['t_ma20/min10'] = data_1['t_ma20']/data_1['t_min10']
data_1['t_ma20/max10'] = data_1['t_ma20']/data_1['t_max10']
data_1['t_ma10_maior_max20'] = np.where((data_1.t_ma10 > data_1.t_max20),1,0)
data_1['t_ma20_menor_min10'] = np.where((data_1.t_ma20 < data_1.t_min10),1,0)
data_1['t_ma10_menor_min20'] = np.where((data_1.t_ma10 < data_1.t_min20),1,0)

# comparações entre ponto atual e máximos e mínimos de 10 e 20 minutos
data_1['t_valor_menor_min20'] = np.where((data_1.TICKVOL < data_1.t_min20),1,0)
data_1['t_valor_menor_min10'] = np.where((data_1.TICKVOL < data_1.t_min10),1,0)
data_1['t_valor_maior_max20'] = np.where((data_1.TICKVOL > data_1.t_max20),1,0)
data_1['t_valor_maior_max10'] = np.where((data_1.TICKVOL > data_1.t_max10),1,0)

# criação das bollinger bands considerando n = 20 minutos e 2 desvios padrão
indicator_aa = BollingerBands(close=data_1["TICKVOL"], n=20, ndev=2)
data_1['t_bb_bbp'] = indicator_aa.bollinger_pband()
data_1['t_bb_upper'] = indicator_aa.bollinger_hband_indicator()
data_1['t_bb_lower'] = indicator_aa.bollinger_lband_indicator()

# criação do tickvol lagado de até 3 minutos
data_1['t_valor_l1'] = data_1['TICKVOL'].shift(1)
data_1['t_valor_l2'] = data_1['TICKVOL'].shift(2)
data_1['t_valor_l3'] = data_1['TICKVOL'].shift(3)
# comparação entre o tickvol atual e o tickvol lagado
data_1['t_diff_l1'] = data_1['TICKVOL']  - data_1['t_valor_l1']
data_1['t_diff_l2'] = data_1['TICKVOL']  - data_1['t_valor_l2']
data_1['t_diff_l3'] = data_1['TICKVOL']  - data_1['t_valor_l3']
# comparação entre o tickvol lagado de 1 minuto e tickvol lagado de 20 minutos
data_1['t_lag1/lag20'] = data_1['TICKVOL'].shift(1)/data_1['TICKVOL'].shift(20)
data_1['t_lag1/lag10'] = data_1['TICKVOL'].shift(1)/data_1['TICKVOL'].shift(10)
data_1['t_coef_ang20'] = (data_1['t_valor_l1'] - data_1['TICKVOL'].shift(20))/20
data_1['t_coef_ang10'] = (data_1['t_valor_l1'] - data_1['TICKVOL'].shift(10))/10

# quantidade de vezes que o tickvol fechou positivo ou negativo em relação ao minuto anteri
# nos últimos 20 minutos e 10 minutos
c = 0
d = 0
for i in range(1,19):
    c = np.where((data_1['TICKVOL'].shift(i) > data_1['TICKVOL'].shift(i+1)),1,0) + c
    d = np.where((data_1['TICKVOL'].shift(i) < data_1['TICKVOL'].shift(i+1)),1,0) + d

e = 0
f = 0
for i in range(1,9):
    e = np.where((data_1['TICKVOL'].shift(i) > data_1['TICKVOL'].shift(i+1)),1,0) + e
    f = np.where((data_1['TICKVOL'].shift(i) < data_1['TICKVOL'].shift(i+1)),1,0) + f

data_1['t_qtde_vezes_positivo20'] = c
data_1['t_qtde_vezes_negativo20'] = d
data_1['t_qtde_vezes_positivo10'] = e
data_1['t_qtde_vezes_negativo10'] = f

# variáveis lagadas vs mínimos, média móvel
data_1['t_lag1/min20'] = data_1['TICKVOL'].shift(1)/data_1['t_min20']
data_1['t_lag2/min20'] = data_1['TICKVOL'].shift(2)/data_1['t_min20']
data_1['t_lag3/min20'] = data_1['TICKVOL'].shift(3)/data_1['t_min20']
data_1['t_lag1/ma20'] = data_1['TICKVOL'].shift(1)/data_1['t_ma20']
```

```
# variável média vs máximo
data_1['t_ma3/max3'] = a.rolling(3).mean()/a.rolling(3).max()

# normalizando
media = data_1['t_ma20'].mean()
data_1['t_ma20'] = data_1['t_ma20']/media
media = data_1['t_ma10'].mean()
data_1['t_ma10'] = data_1['t_ma10']/media

media = data_1['t_max20'].mean()
data_1['t_max20'] = data_1['t_max20']/media
media = data_1['t_max10'].mean()
data_1['t_max10'] = data_1['t_max10']/media
media = data_1['t_min20'].mean()
data_1['t_min20'] = data_1['t_min20']/media
media = data_1['t_min10'].mean()
data_1['t_min10'] = data_1['t_min10']/media

media = data_1['t_bb_upper'].mean()
data_1['t_bb_upper'] = data_1['t_bb_upper']/media
media = data_1['t_bb_lower'].mean()
data_1['t_bb_lower'] = data_1['t_bb_lower']/media

media = data_1['t_valor_l1'].mean()
data_1['t_valor_l1'] = data_1['t_valor_l1']/media
media = data_1['t_valor_l2'].mean()
data_1['t_valor_l2'] = data_1['t_valor_l2']/media
media = data_1['t_valor_l3'].mean()
data_1['t_valor_l3'] = data_1['t_valor_l3']/media

media = data_1['t_diff_l1'].mean()
data_1['t_diff_l1'] = data_1['t_diff_l1']/media
media = data_1['t_diff_l2'].mean()
data_1['t_diff_l2'] = data_1['t_diff_l2']/media
media = data_1['t_diff_l3'].mean()
data_1['t_diff_l3'] = data_1['t_diff_l3']/media

media = data_1['t_coef_ang20'].mean()
data_1['t_coef_ang20'] = data_1['t_coef_ang20']/media
media = data_1['t_coef_ang10'].mean()
data_1['t_coef_ang10'] = data_1['t_coef_ang10']/media

media = data_1['t_qtde_vezes_positivo20'].mean()
data_1['t_qtde_vezes_positivo20'] = data_1['t_qtde_vezes_positivo20']/media
media = data_1['t_qtde_vezes_negativo20'].mean()
data_1['t_qtde_vezes_negativo20'] = data_1['t_qtde_vezes_negativo20']/media
media = data_1['t_qtde_vezes_positivo10'].mean()
data_1['t_qtde_vezes_positivo10'] = data_1['t_qtde_vezes_positivo10']/media
media = data_1['t_qtde_vezes_negativo10'].mean()
data_1['t_qtde_vezes_negativo10'] = data_1['t_qtde_vezes_negativo10']/media

media = data_1['TICKVOL'].mean()
data_1['TICKVOL'] = data_1['TICKVOL']/media

######################
### target lagado ###
######################
data_1['valor_target_l1'] = data_1['target'].shift(1)
data_1['valor_target_l2'] = data_1['target'].shift(2)
data_1['valor_target_l3'] = data_1['target'].shift(3)
data_1['valor_target_l4'] = data_1['target'].shift(4)
```

```
data_1['valor_target_l5'] = data_1['target'].shift(5)
data_1['valor_target_l6'] = data_1['target'].shift(6)
data_1['valor_target_l7'] = data_1['target'].shift(7)
data_1['valor_target_l8'] = data_1['target'].shift(8)
data_1['valor_target_l9'] = data_1['target'].shift(9)
data_1['valor_target_l10'] = data_1['target'].shift(10)
data_1['valor_target_l11'] = data_1['target'].shift(11)
data_1['valor_target_l12'] = data_1['target'].shift(12)
data_1['valor_target_l13'] = data_1['target'].shift(13)
data_1['valor_target_l14'] = data_1['target'].shift(14)
data_1['valor_target_l15'] = data_1['target'].shift(15)
data_1['valor_target_l16'] = data_1['target'].shift(16)
data_1['valor_target_l17'] = data_1['target'].shift(17)
data_1['valor_target_l18'] = data_1['target'].shift(18)
data_1['valor_target_l19'] = data_1['target'].shift(19)
data_1['valor_target_l20'] = data_1['target'].shift(20)

###################################
### variáveis baseadas em datas ###
###################################
# dia da semana
data_1['dia_semana'] = pd.DatetimeIndex(data_1['datetime']).weekday # 0 é segunda #
# hora do dia
data_1['hora_do_dia'] = pd.DatetimeIndex(data_1['datetime']).hour
# período do mês, inicial, meio ou final
data_1['dia_do_mes'] = np.where((pd.DatetimeIndex(data_1['datetime']).day < 10),1,
                                np.where((pd.DatetimeIndex(data_1['datetime']).day < 20),2,3
# período do dia
data_1['periodo_dia'] = np.where((pd.DatetimeIndex(data_1['datetime']).hour < 12),1,
                                np.where((pd.DatetimeIndex(data_1['datetime']).hour < 15),2,

################
### ajustes ###
################
# retira observações vazias
data_1 = data_1.dropna()
data_1.reset_index(drop = True, inplace = True)

data_1
```

Out[2]:

target	c_ma20	c_ma10	c_ma10/ma20	c_ma20_maior_ma10	c_max20	c_max10	...	valor_targ
0	1.137225	1.137527	1.000274	0	1.135332	1.136222	...	
0	1.137302	1.137517	1.000198	0	1.135332	1.136222	...	
1	1.137319	1.137492	1.000160	0	1.135332	1.136222	...	
0	1.137348	1.137487	1.000130	0	1.135332	1.136222	...	
0	1.137358	1.137466	1.000103	0	1.135332	1.136222	...	

target	c_ma20	c_ma10	c_ma10/ma20	c_ma20_maior_ma10	c_max20	c_max10	...	valor_targ
...	...	...	...	...	...	...	...	
0	1.020441	1.020558	1.000123	0	1.019355	1.020154	...	
1	1.020398	1.020558	1.000165	0	1.019355	1.020154	...	
0	1.020372	1.020543	1.000175	0	1.019355	1.020154	...	
1	1.020318	1.020481	1.000168	0	1.019355	1.020154	...	
0	1.020301	1.020446	1.000150	0	1.019355	1.020154	...	

In [3]:

```
treino_1, teste_1 = split_treino_test(data_1)

data_1X = treino_1.drop(columns=['target', 'datetime'])
data_1X

data_1y = treino_1[['target']]
data_1y

X_teste_1 = teste_1.copy().drop(columns = ['target', "datetime"])
y_teste_1 = teste_1[['target']]
X_teste_1
```

Out[3]:

	valor	TICKVOL	c_ma20	c_ma10	c_ma10/ma20	c_ma20_maior_ma10	c_max20	c
72733	0.967724	1.857261	0.972814	0.972312	0.999493	1	0.971699	0
72734	0.968593	2.116299	0.972571	0.971873	0.999290	1	0.971699	0
72735	0.968031	1.920788	0.972287	0.971356	0.999051	1	0.971444	0
72736	0.967571	1.186437	0.972006	0.970824	0.998793	1	0.971444	0
72737	0.968031	0.899770	0.971745	0.970344	0.998566	1	0.971444	0
...	...	...	...	...	...	...	...	
96199	1.019978	0.893435	1.020441	1.020558	1.000123	0	1.019355	1
96200	1.019671	0.372895	1.020398	1.020558	1.000165	0	1.019355	1
96201	1.020029	0.199382	1.020372	1.020543	1.000175	0	1.019355	1
96202	1.019467	0.216804	1.020318	1.020481	1.000168	0	1.019355	1
96203	1.019825	0.432199	1.020301	1.020446	1.000150	0	1.019355	1

23471 rows × 128 columns

In [4]:

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error

mod = xgb.XGBRegressor(
    gamma=1,
    learning_rate=0.01,
    max_depth=4,
    n_estimators=1000,
    subsample=0.8,
    random_state=34
)

mod.fit(data_1X, data_1y)
predictions = mod.predict(X_teste_1)
mse = mean_squared_error(y_teste_1, predictions)
print(mse)
```

```
0.2430941527249074
```

In [5]:

```
y_teste_1['y_pred'] = (mod.predict(X_teste_1) > 0.5).astype("int32")
y_teste_1

##########################
### Matriz de confusão ###
##########################

from sklearn.metrics import confusion_matrix

cnf_matrix = confusion_matrix(y_teste_1['target'],y_teste_1['y_pred'])
cnf_matrix
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/ipykernel_launcher.py:
1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  """Entry point for launching an IPython kernel.
```

Out[5]:

```
array([[13356,   191],
       [ 9751,   173]])
```

In [2]:

```
accuracy = (13356+173)/(13356+191+9751+173)
accuracy
```

Out[2]:

```
0.57641344638064
```

In [4]:

```
recall = (13356)/(13356+9751)
recall
```

Out[4]:

```
0.5780066646470766
```

In [5]:

```
precision = (13356)/(13356+191)
precision
```

Out[5]:

```
0.9859009374769322
```

In [9]:

```
f_score =2*((0.9859*0.5780)/(0.9859+0.5780))
f_score
```

Out[9]:

```
0.7287552912590319
```

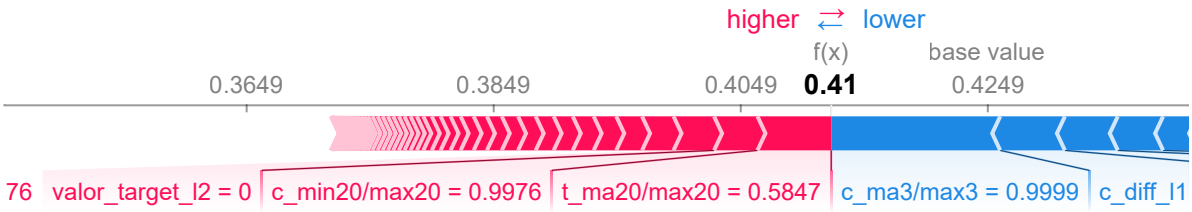
In [6]:

```
import shap
import numpy
shap.initjs()

# explain the model's predictions using SHAP
# (same syntax works for LightGBM, CatBoost, scikit-learn and spark models)
explainer = shap.TreeExplainer(mod)
shap_values = explainer.shap_values(X_teste_1)

id_analysis = 3000
# visualize the first prediction's explanation (use matplotlib=True to avoid Javascript)
shap.force_plot(explainer.expected_value, shap_values[id_analysis,:], X_teste_1.iloc[id_ana
```

Out[6]:



Com base nos 4 modelos desenvolidos, o XGBOOST obteve o melhor resultado com 0,24 de mse (erro predição), na qual sua matriz de confusão apresentou 13356 "menos de 10" e 173 "mais que 10", ambos corretamente. Porém, apresentou 9751 "menor que 10" e 191 "maior que 10" erroneamente. A partir disso, o SHAP apresentou que a feature que mais impacta na predição é a "c_ma3/ma3" contribuindo 0,149 no valor base da predição. Resultando então, com precisão de 0.98 e f-score (equílibrio entre acurácia e recall) de 0.73.

A conclusão deste trabalho é que o modelo de árvore (XGBOOST) teve um desempenho superior ao modelo simples (naive) e a rede neural artificial, sendo recomendado nesse cenário.

Anexo: Código da Rede Neural Artifical desenvolvida como comparação aos outros modelos.

---------- split dos dados para validação cruzada ----------

```
from sklearn.model_selection import TimeSeriesSplit
```

treino_1, teste_1 = split_treino_test(data_1)

```
data_1X = treino_1.drop(columns=['target', 'datetime']) data_1X
```

```
data_1y = treino_1[['target']] data_1y
```

```
tscv = TimeSeriesSplit() print(tscv) TimeSeriesSplit(gap=0,
max_train_size=None, n_splits=5, test_size=None) for
train_index, test_index in tscv.split(treino_1): ... print("TRAIN:",
train_index, "TEST:", test_index) ... X_teste_1=
data_1X.loc[test_index] ... X_treino_1= data_1X.loc[train_index]
... y_treino_1 = data_1y.loc[train_index] ... y_teste_1 =
data_1y.loc[test_index]
```

... X_treino_1['testeR']=train_index ... y_treino_1['testeR']=train_index ... X_teste_1['testeR']=test_index ... y_teste_1['testeR']=test_index

out: TimeSeriesSplit(gap=0, max_train_size=None, n_splits=5, test_size=None) TRAIN: [0 1 2 ... 12120 12121 12122] TEST: [12123 12124 12125 ... 24242 24243 24244] TRAIN: [0 1 2 ... 24242 24243 24244] TEST: [24245 24246 24247 ... 36364 36365 36366] TRAIN: [0 1 2 ... 36364 36365 36366] TEST: [36367 36368 36369 ... 48486 48487 48488] TRAIN: [0 1 2 ... 48486 48487 48488] TEST: [48489 48490 48491 ... 60608 60609 60610] TRAIN: [0 1 2 ... 60608 60609 60610] TEST: [60611 60612 60613 ... 72730 72731 72732]

---------- criação dos folds para treino e teste ----------

X_treino_cv1 = X_treino_1[X_treino_1['testeR'] <= 12122] X_treino_cv1 = X_treino_cv1.copy().drop(columns = ['testeR']) X_teste_cv1 = X_treino_1[X_treino_1['testeR'] >= 12123] X_teste_cv1 = X_teste_cv1[X_teste_cv1['testeR'] <= 24244] X_teste_cv1 = X_teste_cv1.copy().drop(columns = ['testeR'])

X_treino_cv2 = X_treino_1[X_treino_1['testeR'] <= 24244] X_treino_cv2 = X_treino_cv2.copy().drop(columns = ['testeR']) X_teste_cv2 = X_treino_1[X_treino_1['testeR'] >= 24245] X_teste_cv2 = X_teste_cv2[X_teste_cv2['testeR'] <= 36366] X_teste_cv2 = X_teste_cv2.copy().drop(columns = ['testeR'])

X_treino_cv3 = X_treino_1[X_treino_1['testeR'] <= 36366] X_treino_cv3 = X_treino_cv3.copy().drop(columns = ['testeR']) X_teste_cv3 = X_treino_1[X_treino_1['testeR'] >= 36367] X_teste_cv3 = X_teste_cv3[X_teste_cv3['testeR'] <= 48488] X_teste_cv3 = X_teste_cv3.copy().drop(columns = ['testeR'])

X_treino_cv4 = X_treino_1[X_treino_1['testeR'] <= 48488] X_treino_cv4 = X_treino_cv4.copy().drop(columns = ['testeR']) X_teste_cv4 = X_treino_1[X_treino_1['testeR'] >= 48489] X_teste_cv4 = X_teste_cv4[X_teste_cv4['testeR'] <= 60610] X_teste_cv4 = X_teste_cv4.copy().drop(columns = ['testeR'])

X_treino_cv5 = X_treino_1.copy().drop(columns = ['testeR']) X_teste_cv5 = X_teste_1.copy().drop(columns = ['testeR'])

y_treino_cv1 = y_treino_1[y_treino_1['testeR'] <= 12122] y_treino_cv1 = y_treino_cv1.copy().drop(columns = ['testeR']) y_teste_cv1 = y_treino_1[y_treino_1['testeR'] >= 12123] y_teste_cv1 = y_teste_cv1[y_teste_cv1['testeR'] <= 24244] y_teste_cv1 = y_teste_cv1.copy().drop(columns = ['testeR'])

y_treino_cv2 = y_treino_1[y_treino_1['testeR'] <= 24244] y_treino_cv2 = y_treino_cv2.copy().drop(columns = ['testeR']) y_teste_cv2 = y_treino_1[y_treino_1['testeR'] >= 24245] y_teste_cv2 = y_teste_cv2[y_teste_cv2['testeR'] <= 36366] y_teste_cv2 = y_teste_cv2.copy().drop(columns = ['testeR'])

y_treino_cv3 = y_treino_1[y_treino_1['testeR'] <= 36366] y_treino_cv3 = y_treino_cv3.copy().drop(columns = ['testeR']) y_teste_cv3 = y_treino_1[y_treino_1['testeR'] >= 36367] y_teste_cv3 = y_teste_cv3[y_teste_cv3['testeR'] <= 48488] y_teste_cv3 = y_teste_cv3.copy().drop(columns = ['testeR'])

y_treino_cv4 = y_treino_1[y_treino_1['testeR'] <= 48488] y_treino_cv4 = y_treino_cv4.copy().drop(columns = ['testeR']) y_teste_cv4 = y_treino_1[y_treino_1['testeR'] >= 48489] y_teste_cv4 = y_teste_cv4[y_teste_cv4['testeR'] <= 60610] y_teste_cv4 = y_teste_cv4.copy().drop(columns = ['testeR'])

y_treino_cv5 = y_treino_1.copy().drop(columns = ['testeR']) y_teste_cv5 = y_teste_1.copy().drop(columns = ['testeR'])

##

Criação da estrutura da Rede Neural

modelo = keras.Sequential()

criação da camada de entrada, ativação relu

modelo.add(keras.layers.Dense(128, activation='relu', input_shape=(128,)))

criação da camada escondida com 3 neurônios, ativação relu

modelo.add(keras.layers.Dense(3, activation='relu'))

criação da camada de saída, ativação sigmóide para saída em probabilidade

modelo.add(keras.layers.Dense(1, activation='sigmoid'))

alteração do tamanho do learning rate do otimizador SGD

sgd = SGD(lr=0.00001, decay=1e-6, momentum=0.9, nesterov=True) modelo.compile(optimizer=sgd, loss='binary_crossentropy', metrics=['accuracy'])

##

cv1

historico1 = modelo.fit(x=X_treino_cv1, y=y_treino_cv1, epochs=10, batch_size=64, verbose = 0, validation_data=(X_teste_cv1, y_teste_cv1)) print('treino k-fold1') score_n1 = modelo.evaluate(X_treino_cv1, y_treino_cv1, verbose=0) print(score_n1) print('teste validação k-fold1') score_n1_t = modelo.evaluate(X_teste_cv1, y_teste_cv1, verbose=0) print(score_n1_t)

##

cv2

historico2 = modelo.fit(x=X_treino_cv2, y=y_treino_cv2, epochs=10, batch_size=64, verbose = 0, validation_data=(X_teste_cv2, y_teste_cv2)) print('treino k-fold2') score_n2 = modelo.evaluate(X_treino_cv2, y_treino_cv2, verbose=0) print(score_n2) print('teste validação k-fold2') score_n2_t = modelo.evaluate(X_teste_cv2, y_teste_cv2, verbose=0) print(score_n2_t)

##

cv3

historico3 = modelo.fit(x=X_treino_cv3, y=y_treino_cv3, epochs=10, batch_size=64, verbose = 0, validation_data=(X_teste_cv3, y_teste_cv3)) print('treino k-fold3') score_n3 = modelo.evaluate(X_treino_cv3, y_treino_cv3, verbose=0) print(score_n3) print('teste validação k-fold3') score_n3_t = modelo.evaluate(X_teste_cv3, y_teste_cv3, verbose=0) print(score_n3_t)

##

cv4

historico4 = modelo.fit(x=X_treino_cv4, y=y_treino_cv4, epochs=10, batch_size=64, verbose = 0, validation_data=(X_teste_cv4, y_teste_cv4)) print('treino k-fold4') score_n4 = modelo.evaluate(X_treino_cv4, y_treino_cv4, verbose=0) print(score_n4) print('teste validação k-fold4') score_n4_t = modelo.evaluate(X_teste_cv4, y_teste_cv4, verbose=0) print(score_n4_t)

##

cv5

historico5 = modelo.fit(x=X_treino_cv5, y=y_treino_cv5, epochs=10, batch_size=64, verbose = 0, validation_data=(X_teste_cv5, y_teste_cv5)) print('treino k-fold5') score_n5 = modelo.evaluate(X_treino_cv5, y_treino_cv5, verbose=0) print(score_n5) print('teste validação k-fold5') score_n5_t = modelo.evaluate(X_teste_cv5, y_teste_cv5, verbose=0) print(score_n5_t)

avaliação da acurácia nos k-folds

print('Acurácia - Teste Validação') (score_n1_t[1]+score_n2_t[1]+score_n3_t[1]+score_n4_t[1]+score_n5_t[1])/5

treino k-fold1 [0.6902827620506287, 0.6262476444244385] teste validação k-fold1 [0.6909699440002441, 0.5966837406158447] treino k-fold2 [0.6867834329605103, 0.6114662885665894] teste validação k-fold2 [0.6897225379943848, 0.5635208487510681] treino k-fold3 [0.6845958232879639, 0.5954849123954773] teste validação k-fold3 [0.6901473999023438, 0.5419072508811951] treino k-fold4 [0.6840112805366516, 0.5820907950401306] teste validação k-fold4 [0.6872687339782715, 0.5591486692428589] treino k-fold5 [0.6833534836769104, 0.577502429485321] teste validação k-fold5 [0.6858381032943726, 0.563438355922699] Acurácia - Teste Validação Out: 0.5649397730827331

################################

Criação do modelo final

################################

X_treino_final = data_1X y_treino_final = data_1y

Criação da estrutura da Rede Neural

modelo = keras.Sequential()

criação da camada de entrada, ativação relu

modelo.add(keras.layers.Dense(128, activation='relu', input_shape=(128,)))

criação da camada escondida com 3 neurônios, ativação relu

modelo.add(keras.layers.Dense(3, activation='relu'))

criação da camada de saída, ativação sigmóide para saída em probabilidade

modelo.add(keras.layers.Dense(1, activation='sigmoid'))

alteração do tamanho do learning rate do otimizador SGD

sgd = SGD(lr=0.00001, decay=1e-6, momentum=0.9, nesterov=True) modelo.compile(optimizer=sgd, loss='binary_crossentropy', metrics=['accuracy'])

treino final

print('treino final') historico1 = modelo.fit(x=X_treino_final, y=y_treino_final, epochs=10, batch_size=64)

Out:
treino final Epoch 1/10 1137/1137 [==============================] - 7s 7ms/step - loss: 19.4732 - accuracy: 0.5749A: 0s - loss: 22.2951 - accuracy - ETA: 0s - loss: 20.9616 - acc Epoch 2/10 1137/1137 [==============================] - 7s 6ms/step - loss: 0.6922 - accuracy: 0.5752: Epoch 3/10

1137/1137 [==============================] - 6s 5ms/step - loss: 0.6917 - accuracy: 0.5752 Epoch 4/10
1137/1137 [==============================] - 5s 5ms/step - loss: 0.6911 - accuracy: 0.5752 Epoch 5/10
1137/1137 [==============================] - 6s 5ms/step - loss: 0.6906 - accuracy: 0.5752 Epoch 6/10
1137/1137 [==============================] - 6s 5ms/step - loss: 0.6901 - accuracy: 0.5752 Epoch 7/10
1137/1137 [==============================] - 6s 5ms/step - loss: 0.6897 - accuracy: 0.5752 Epoch 8/10
1137/1137 [==============================] - 5s 5ms/step - loss: 0.6892 - accuracy: 0.5752 Epoch 9/10
1137/1137 [==============================] - 7s 6ms/step - loss: 0.6888 - accuracy: 0.5752 Epoch 10/10 1137/1137 [==============================] - 7s 6ms/step - loss: 0.6885 - accuracy: 0.5752

##################################

Avaliação na amostra teste

##################################

X_teste_1 = teste_1.copy().drop(columns = ['target', "datetime"]) y_teste_1 = teste_1[['target']]

avaliação na amostra teste

score_n = modelo.evaluate(X_teste_1, y_teste_1, verbose=0) print('resultado na amostra de teste') score_n

resultado na amostra de teste Out: [0.6881194114685059, 0.5771803259849548]

y_teste_1['y_pred'] = (modelo.predict(X_teste_1) > 0.5).astype("int32") y_teste_1

#########################

Matriz de confusão

#########################

from sklearn.metrics import confusion_matrix

cnf_matrix = confusion_matrix(y_teste_1['target'],y_teste_1['y_pred']) cnf_matrix

out> ([[13547, 0], [9924, 0]])