

UNIVERSIDADE DE SÃO PAULO
Instituto de Instituto de Ciências Matemáticas e de Computação

Introdução à Computação no Mercado Financeiro

Prof. Denis Fernando Wolf

Trabalho 3

Aluna: Bruna Gongora Bariccatti

Nº USP: 12674933

Data máxima de entrega: 10/07/2023

Objetivos

Com o presente trabalho tenho como objetivo criar dois algoritmos, sendo um de alocação e um de trading sistemático. Para o algoritmo de alocação utilizei a rede neural MLP e para o algoritmo de trading sistemático utilizei a random forest. A escolha desses algoritmos e estratégias de aprendizado de máquinas foi feito para englobar a maior parte possível do conteúdo visto nas aulas gravadas e discutido nas aulas online.

Métodos Utilizados

Em meu algoritmo de alocação sistemática usando redes neurais primeiramente escolhi os intervalos de cálculos de momentos que irei utilizar para calcular meus valores de entrada. Acabei optando pelos intervalos de 1, 6 e 12 meses, pois analisando outros intervalos esse foi o que obtive melhores resultados. Para a escolha de ativos utilizei o IMAB como base de dados no cálculo de momento e também o IBOV para o treinamento do algoritmo. Para os parâmetros da função MLPClassifier acabei optando pelo solver = 'lbfgs', já que é o mais adequado quando há um conjunto de dados pequenos, como é o meu caso. No activation utilizei o 'tanh', que utiliza a função hiperbólica da tangente para otimizações. Dentre as possíveis escolhas ('identity','logistic','tanh','relu'), foi a que melhor apresentou resultados nos testes. Defino também o warm_start = True e o early_stopping = True para utilizar funções da chamada anterior como iniciação e para realizar uma parada antecipada se os resultados não estiverem melhorando.

Já para o trading sistemático utilizando random forest primeiramente escolhi uma ação para ser aplicada o trading. Escolhi as ações da empresa Totvs, que atua com softwares de gestão empresarial para diferentes negócios, como: agroindústria, construção, educacional, serviços financeiros, manufatura. Para os parâmetros da Random Forest utilizei 20 árvores de decisões, com uma profundidade máxima de 20 e realizo a análise dos subconjuntos com o auxílio da função log2, ao invés do padrão de análise, a raiz quadrada.

Procedimento e Resultados

Alocação Sistemática Usando Rede Neural

Para começar o programa importo o modelo de otimização da biblioteca Scikit Learn e também funções para métrica e classificações. Também importo bibliotecas necessárias para a realização de cálculos durante o código, sendo essas: numpy, pandas, matplotlib e seaborn.

```
[181] #Importando o modelo otimiza a função log-loss
      from sklearn.neural_network import MLPClassifier
      #Importando funções de métricas, como funções de perda, pontuação e utilidades
      from sklearn import metrics

      import numpy as np
      import pandas as pd
      import matplotlib.pyplot as plt
      import seaborn as sns

      sns.set(style='whitegrid')
```

Realizo a leitura dos dados fornecidos pelo professor, armazenando esses.

```
[182] dados=pd.read_excel('Dados_Classes_Indices.xlsx', engine='openpyxl')
      dados.set_index(keys = 'Data', inplace = True)
```

Calculo as variações parciais mensais para os dados presentes no Excel.

```
[183] dados_chg = dados.pct_change()
      dados_chg.fillna(0, inplace=True)
```

Aqui defino funções para realizar o cálculo de momento.

```
[576] #Definição para cálculo de momento em 1 mês
      dados_mom1 = dados.copy()
      dados_mom1.iloc[0:3] = 0
      mom_period = 1

      for ind in range(mom_period, len(dados.index)):
          dados_mom1.iloc[ind] = dados.iloc[ind]/dados.iloc[ind-mom_period]

      #Definição para cálculo de momento em 6 meses
      dados_mom6 = dados.copy()
      dados_mom6.iloc[0:6] = 0
      mom_period = 6

      for ind in range(mom_period, len(dados.index)):
          dados_mom6.iloc[ind] = dados.iloc[ind]/dados.iloc[ind-mom_period]

      #Definição para cálculo de momento em 12 meses
      dados_mom12 = dados.copy()
      dados_mom12.iloc[0:12] = 0
      mom_period = 12

      for ind in range(mom_period, len(dados.index)):
          dados_mom12.iloc[ind] = dados.iloc[ind]/dados.iloc[ind-mom_period]
```

Para a alocação em classes usando redes neurais utilizei como valores de entrada o momento do IMAB. Já para os intervalos do momento utilizei intervalos diferentes dos escolhidos pelo professor, sendo esses: um mês, seis meses e 12 meses.

Nesse ponto crio um data frame com os dados que utilizarei para ensinar a minha rede neural. Analiso em cada um dos meses qual ativo desempenhou um melhor papel, o que tiver o melhor desempenho assumo que é o ativo que gostaria de estar investindo.

```
[577] dados_apr = dados_chg[['IMAB', 'IBOV']].copy()

      dados_apr['MOM1'] = dados_mom1['IMAB']
      dados_apr['MOM6'] = dados_mom6['IMAB']
      dados_apr['MOM12'] = dados_mom12['IMAB']

      dados_apr['IMAB-BUY'] = np.argmin(dados_apr[['IMAB', 'IBOV']]
                                         .reset_index().drop(['Data'], axis=1).to_numpy(), axis=1)
      dados_apr['IBOV-BUY'] = np.argmax(dados_apr[['IMAB', 'IBOV']]
                                         .reset_index().drop(['Data'], axis=1).to_numpy(), axis=1)
```

Crio dados de entrada e saída, entrada sendo os momentos do IMAB em 1, 3, 12 meses e saída os dados definidos anteriormente como IMAB-BUY e IBOV-BUY.

```
[578] din = dados_apr[['MOM1', 'MOM6', 'MOM12']].reset_index().drop(['Data'], axis=1).to_numpy()
      dout = dados_apr[['IMAB-BUY', 'IBOV-BUY']].reset_index().drop(['Data'], axis=1).to_numpy()

      print("Data samples:", dout.shape[0])

Data samples: 236
```

Escolho o número de casos que irei utilizar para teste e os que irei utilizar para validação. Feita essa escolha divido o conjunto em entradas e saídas.

```
[579] n_train = 100 #Número de casos para treino

      train_in = din[12:12+n_train] #Entrada para treino
      train_out = dout[13:13+n_train] #Valor de validação do treino

      val_in = din[12+n_train:dout.shape[0]-1]
      val_out = dout[13+n_train:dout.shape[0]]
```

Nesse passo importo a biblioteca de redes neurais e crio uma rede por meio da função MLPClassifier. Defino random state como um inteiro, no caso 1, para reproduzir os resultados mesmo chamando a função novamente. Determino o número de neurônios como 20 e o número máximo de iterações como 100000. Já para a escolha do solver utilizei o 'lbfgs' já que, de acordo com a própria documentação da biblioteca, é o mais apropriado para conjunto de dados pequenos, já que converge mais rápido e tem um melhor desempenho.

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(random_state=1, hidden_layer_sizes=(20, ), max_iter=100000, solver='lbfgs',
activation='tanh', alpha=0.0001, warm_start=True, early_stopping=True)

clf.fit(train_in, train_out)
```

MLPClassifier

MLPClassifier(activation='tanh', early_stopping=True, hidden_layer_sizes=(20,), max_iter=100000, random_state=1, solver='lbfgs', warm_start=True)

Nesse ponto avalio os resultados encontrados. Baseado nos dados de treinamento, prevejo a saída e comparo esse valor aos valores desejados de saída, ou seja, comparo a saída do algoritmo com a saída desejada. Já na segunda análise comparo os valores encontrados pelo algoritmo com os valores de validação, esses sendo que esses valores não foram utilizados para o treinamento.

```
y_pred = clf.predict(train_in)
print("Precisão do treinamento:", metrics.accuracy_score(train_out, y_pred))

y_pred = clf.predict(val_in)
print("Precisão dos testes de validação:", metrics.accuracy_score(val_out, y_pred))

y_pred = clf.predict(din)

Precisão do treinamento: 0.64
Precisão dos testes de validação: 0.5447154471544715
```

Para a análise financeira dos resultados realizo os seguintes cálculos, transformando os resultados do meu algoritmo em duas colunas do data frame criado anteriormente.

```
dados_apr['IMAB-BUY-APR'] = np.argmax(y_pred, axis=1)
dados_apr['IBOV-BUY-APR'] = np.argmax(y_pred, axis=1)

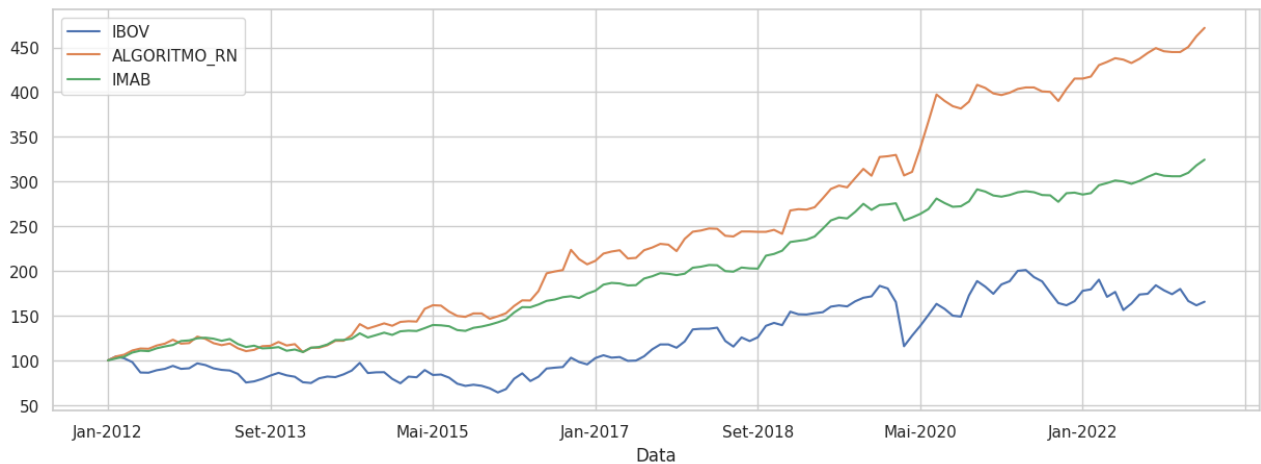
dados_apr['IMAB-BUY-APR'] = dados_apr['IMAB-BUY-APR'].shift(1)
dados_apr['IBOV-BUY-APR'] = dados_apr['IBOV-BUY-APR'].shift(1)

dados_apr['APR-CHG'] = (dados_apr['IMAB'] * dados_apr['IMAB-BUY-APR']) + (dados_apr['IBOV'] * dados_apr['IBOV-BUY-APR'])
dados['APR-ACC'] = (1 + dados_apr['APR-CHG']).cumprod()
```

Calculado os dados do modelo, crio uma função para gerar um gráfico apresentando os valores das ações de entrada, que em meu modelo é o IBOV e o IMAB, juntamente com o algoritmo de alocação que criei.

```
dados = dados*100 / dados.iloc[n_train]
dados[['IBOV', 'APR-ACC', 'IMAB']].iloc[n_train:].plot(figsize = (15,5))
```

Vemos pelo gráfico que os valores encontrados pelo algoritmo criado apresentam melhores resultados que o IBOV e o IMAB. Entretanto, essa diferença é pequena, tentei aumentá-lo modificando os parâmetros, mas não consegui. Acredito que o motivo da dificuldade de implementar um bom modelo seja os valores de entrada.



Aqui realizo os cálculos necessários para visualizar os valores matemáticos de retorno acumulado, anualizado e a volatilidade anualizada.

```
ref_data = n_train
periodo = int((len(dados.index[ref_data+1:])/12))
print("Período:", dados.index[ref_data+1], "-", dados.index[-1], '(', periodo, ')')

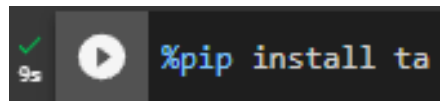
ret_acc = (dados[['IMAB', 'IBOV', 'ALGORITMO_RN']].iloc[-1]/dados[['IMAB', 'IBOV', 'ALGORITMO_RN']].iloc[ref_data])
print("Retorno acumulado:\n", ret_acc)
ret_aa = ((dados[['IMAB', 'IBOV', 'ALGORITMO_RN']].iloc[-1]/dados[['IMAB', 'IBOV', 'ALGORITMO_RN']].iloc[ref_data])**((1/periodo)))-1
print("Retorno anualizado:\n", ret_aa)
vol_aa = dados_apr[['IMAB', 'IBOV', 'ALGORITMO_V']].iloc[ref_data+1:].std()*np.sqrt(12)
print("Vol anualizada:\n", vol_aa)
```

Podemos ver que tanto o retorno acumulado quanto o anualizado do algoritmo são maiores que os ativos utilizados como comparação.

```
Período: Fev-2012 - Abr-2023 ( 11 )
Retorno acumulado:
IMAB          3.244955
IBOV          1.655753
ALGORITMO_RN  4.718897
dtype: float64
Retorno anualizado:
IMAB          0.112945
IBOV          0.046908
ALGORITMO_RN  0.151485
dtype: float64
Vol anualizada:
IMAB          0.073713
IBOV          0.221852
ALGORITMO_V   0.117203
dtype: float64
```

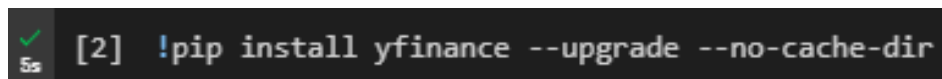
Trading Sistemático usando Random Forest

Realizo a instalação do pacote de análise técnica.



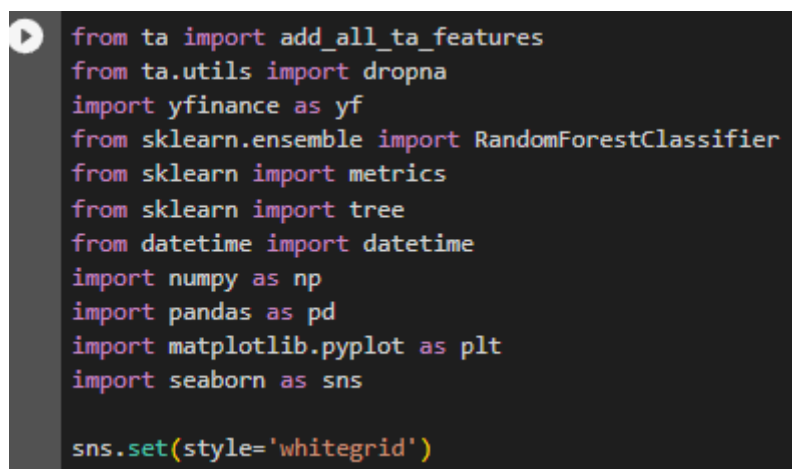
```
%pip install ta
```

Também instalo o pacote do Yahoo Finance .



```
[2] !pip install yfinance --upgrade --no-cache-dir
```

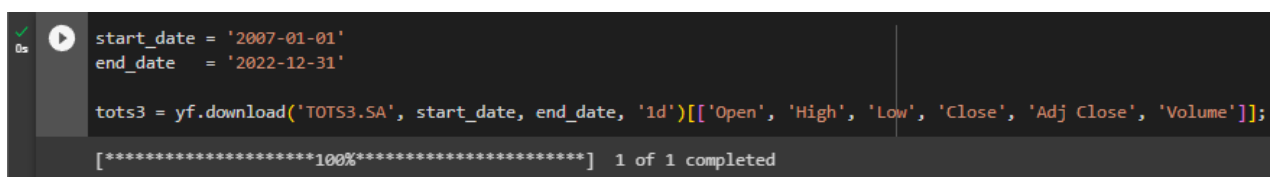
Importo bibliotecas que irei utilizar ao longo de meu código.



```
from ta import add_all_ta_features
from ta.utils import dropna
import yfinance as yf
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn import tree
from datetime import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style='whitegrid')
```

Defino o intervalo de análise de dados da minha ação. Como a ação escolhida por mim entrou no mercado em 2006 comecei minha análise de dados somente em 2007 até o ano de 2022.

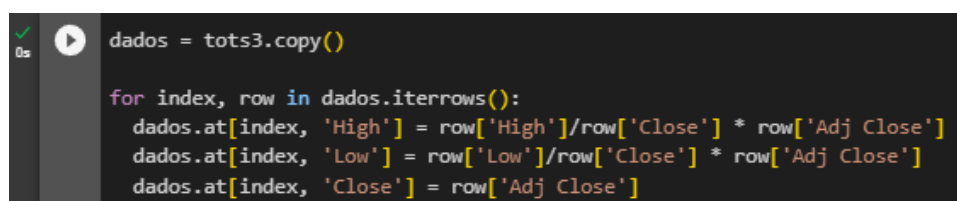


```
start_date = '2007-01-01'
end_date = '2022-12-31'

tots3 = yf.download('TOTS3.SA', start_date, end_date, '1d')[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']];

[*****100%*****] 1 of 1 completed
```

Baseado no fechamento ajustado, ajusto as máximas e mínimas para utilização desses dados a diante.



```
dados = tots3.copy()

for index, row in dados.iterrows():
    dados.at[index, 'High'] = row['High']/row['Close'] * row['Adj Close']
    dados.at[index, 'Low'] = row['Low']/row['Close'] * row['Adj Close']
    dados.at[index, 'Close'] = row['Adj Close']
```


Utilizei o mesmo algoritmo desenvolvido pelo professor, porém mudo o intervalo das janelas para 8 dias, encontrando pontos de mínimo e de máximo e definindo para o algoritmo que nos pontos de mínimo era pro algoritmo ter comprado a ação e nos pontos de máximo vendido.

```
[53] dados['Action']=-1000

periodo_inicio = 0
periodo_tam = 8
periodo_fim = periodo_inicio + periodo_tam

total_dados = dados.shape[0]
cont = 0

while cont < (total_dados):
    min = dados['Close'].iloc[periodo_inicio:periodo_fim].min()
    max = dados['Close'].iloc[periodo_inicio:periodo_fim].max()
    for i in range(periodo_inicio, periodo_fim+1):
        if ((dados['Close'].iloc[i] == min)):
            dados.loc[dados.index[i], 'Action'] = 0
            dados.loc[dados.index[i+1], 'Action'] = 1
        elif ((dados['Close'].iloc[i] == max)):
            dados.loc[dados.index[i], 'Action'] = 0
            dados.loc[dados.index[i+1], 'Action'] = -1
        elif (dados['Action'].iloc[i] == -1000):
            dados.loc[dados.index[i], 'Action'] = 0
    periodo_inicio = periodo_fim + 1
    periodo_fim = periodo_inicio + periodo_tam
    cont = periodo_fim
```

Calculo aqui as bandas de Bollinger inferiores, superiores e médias, as dividindo para se tornarem relativas ao fechamento.

```
from ta.volatility import BollingerBands
ind_bb = BollingerBands(close=dados['Close'], window=20, window_dev=2)
dados['bb_hb'] = ind_bb.bollinger_hband()
dados['bb_lb'] = ind_bb.bollinger_lband()
dados['bb_mb'] = ind_bb.bollinger_mavg()
dados['bb_hr'] = dados['bb_hb']/dados['Close']
dados['bb_lr'] = dados['bb_lb']/dados['Close']
dados['bb_mr'] = dados['bb_mb']/dados['Close']
```

Importo o RSIIndicator para compara a magnitude dos ganhos e perdas recentes durante o período de tempo especificado para medir a velocidade e a mudança dos movimentos de preços de um título. Também normalizo o valor do RSI.

```
from ta.momentum import RSIIndicator
ind_rsi = RSIIndicator(close=dados['Close'], window=2)
dados['RSI'] = ind_rsi.rsi()/100.0
```

Excluo os 20 primeiros resultados, pois esses não possuem informações suficientes para o cálculo das bandas de Bollinger e armazeno o restante dos dados de trading. Em meu estudo consegui 3927 dados de trading.

```
[12]
dados = dados.iloc[20:]

din = dados[['bb_hr', 'bb_lr', 'bb_mr', 'RSI']].reset_index().drop(['Date'], axis=1).to_numpy()
dout = dados['Action'].reset_index().drop(['Date'], axis=1).to_numpy()

print(din.shape)
print(dout.shape)

(3927, 4)
(3927, 1)
```

Divido os meus dados para que o treino seja feito em 1000 dias e analiso os resultados do meu treinamento no restante dos dias, ou seja, do dia 1001 até o 3500.

```
train_n = 1000

train_in = din[0:train_n]
train_out = dout[1:train_n+1]

test_in = din[train_n :3500]
test_out = dout[train_n+1:3501]
```

Utilizo aqui o Random Forest, um algoritmo de aprendizagem que gera várias árvores de decisão, no meu caso utilizei 20 árvores (`n_estimators=20`), onde as decisões são tomadas por votação uniforme. Além de limitar o número de árvores limitei a profundidade máxima dessas árvores para 20 (`max_depth=20`) e alterei o parâmetro que analisa subconjuntos e tenta achar a melhor divisão para utilizar log2 em sua busca (`max_features='log2'`).

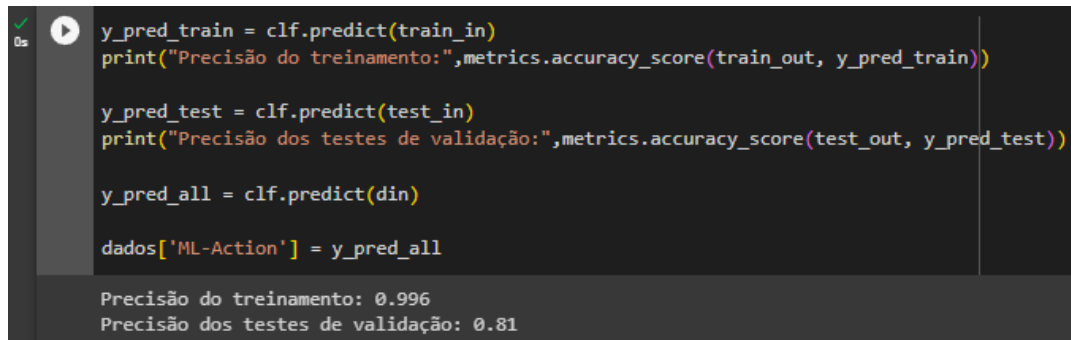
```
[49] #Utilizando a Random Forest para o trading
clf = RandomForestClassifier(random_state=1, max_depth=20, n_estimators=20, max_features='log2')

clf.fit(train_in, train_out.ravel())
```

RandomForestClassifier

RandomForestClassifier(max_depth=20, max_features='log2', n_estimators=20, random_state=1)

Valido os meus resultados de saída, no primeiro analiso os resultados de saída do meu código com a saída ideal estipulada anteriormente, que foi usada para a aprendizagem do código. Já no segundo realizo uma comparação dos resultados de saída do meu código com os resultados de validação.



```
0s y_pred_train = clf.predict(train_in)
print("Precisão do treinamento:", metrics.accuracy_score(train_out, y_pred_train))

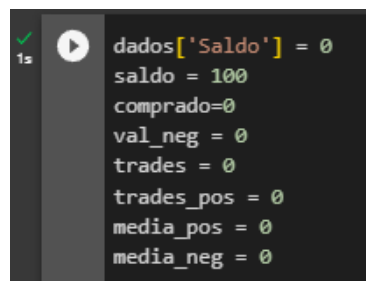
y_pred_test = clf.predict(test_in)
print("Precisão dos testes de validação:", metrics.accuracy_score(test_out, y_pred_test))

y_pred_all = clf.predict(din)

dados['ML-Action'] = y_pred_all

Precisão do treinamento: 0.996
Precisão dos testes de validação: 0.81
```

Defino um código para compra e venda de ações, calculando o total de trades, trades positivos, trades negativos, média dos trades positivos, média dos trades negativos, rentabilidade de comprar a ação e só vende-la ao final do período e rentabilidade das tradings realizadas pelo algoritmo.



```
1s dados['Saldo'] = 0
saldo = 100
comprado = 0
val_neg = 0
trades = 0
trades_pos = 0
media_pos = 0
media_neg = 0
```

```

for i in range(0, dados.shape[0]):
    if (dados['ML-Action'].iloc[i] == 1 and comprado == 0):
        val_neg = dados['Close'].iloc[i]
        comprado = 1
    elif (dados['ML-Action'].iloc[i] == 1 and comprado == -1):
        res_trade = val_neg / dados['Close'].iloc[i] - 1
        if res_trade > 0:
            trades_pos = trades_pos + 1
            media_pos = media_pos + res_trade
        else:
            media_neg = media_neg + res_trade
            saldo = saldo * (1 + res_trade)
            trades = trades + 1
            comprado = 0
    elif (dados['ML-Action'].iloc[i] == -1 and comprado == 0):
        comprado = -1
        val_neg = dados['Close'].iloc[i]
    elif (dados['ML-Action'].iloc[i] == -1 and comprado == 1):
        res_trade = dados['Close'].iloc[i]/val_neg - 1
        if res_trade > 0:
            trades_pos = trades_pos + 1
            media_pos = media_pos + res_trade
        else:
            media_neg = media_neg + res_trade
            saldo = saldo * (1 + res_trade)
            trades = trades + 1
            comprado = 0
    dados.loc[dados.index[i], 'Saldo'] = saldo

```

```

if (trades_pos > 0):
    print("Total de Trades:", trades, "Trades positivos:", trades_pos,
          "(", round(trades_pos/trades*100, 2), "%)")
    print("Média trades positivos:", round(media_pos/trades_pos*100, 2),
          "%, Média trades negativos:", round(media_neg/(trades-trades_pos)*100, 2), "%")
    print('\'Rentabilidade "buy and hold":\'',
          round((dados['Close'].iloc[-1]/dados['Close'].iloc[0]-1)*100, 2), "%")
    print("Rentabilidade trades do algoritmo de aprendizagem:",
          round((dados['Saldo'].iloc[-1]/dados['Saldo'].iloc[0]-1)*100, 2), "%")
else:
    print("Não foram feitos trades no período.")

```

```

Total de Trades: 117 Trades positivos: 102 ( 87.18 %)
Média trades positivos: 9.81 %, Média trades negativos: -8.47 %
Rentabilidade "buy and hold": 1021.35 %
Rentabilidade trades do algoritmo de aprendizagem: 301442.91 %

```

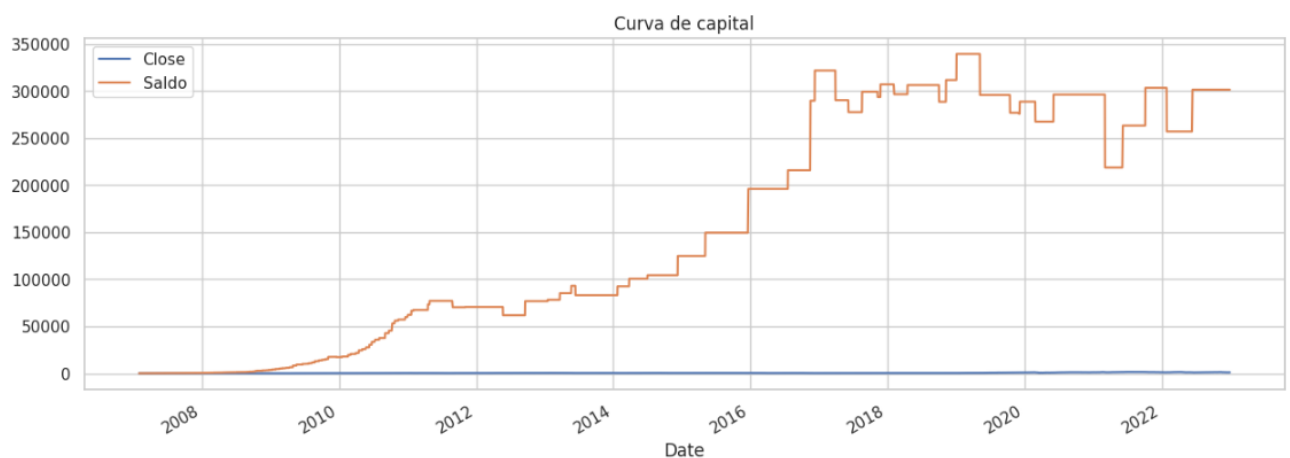
Defino um código para a criação da curva de capital, que mostra o desempenho financeiro do “buy and hold” e dos trading, assumindo nos dois casos que começo com 100 de investimento.

```

[18] dados[['Close', 'Saldo']] = dados[['Close', 'Saldo']] / dados[['Close', 'Saldo']].iloc[0] * 100
      dados[['Close', 'Saldo']].iloc[0:].plot(figsize = (15,5), title='Curva de capital');

```

No gráfico está presente os resultados do meu algoritmo de aprendizagem. Como utilizei os 1000 primeiros dias para treinamento e os outros 2500 para validação temos que aproximadamente o primeiro terço do gráfico mostra os resultados do treinamento do meu código. Já os outros dois terços mostram os resultados da aplicação do código treinado. Temos resultados muito bons de treinamento e também resultados muito bons da aplicação do treinamento até o ano de 2017. Após esse período há uma certa estabilização dos resultados obtidos. Podemos associar esse pior desempenho após o ano de 2017 pois esses valores já estão em uma distância considerável dos valores utilizados para o treinamento.



Aqui temos os resultados mostrados no gráfico e valores que utilizei para a realização dos cálculos.

[] dados																
	Open	High	Low	Close	Adj Close	Volume	Action	bb_hb	bb_lb	bb_mb	bb_hr	bb_lr	bb_mr	RSI	ML-Action	Saldo
Date																
2007-01-31	3.160661	2.471874	2.374399	100.000000	2.444381	1130143	0	2.477060	2.256492	2.366776	1.013369	0.923134	0.968252	0.914216	0	100.000000
2007-02-01	3.327012	2.499368	2.448880	100.613510	2.459377	293055	0	2.479132	2.255419	2.367276	1.008032	0.917069	0.962551	0.932559	0	100.000000
2007-02-02	3.260471	2.449380	2.420887	99.775069	2.438883	126239	0	2.477150	2.256351	2.366751	1.015691	0.925158	0.970424	0.588574	0	100.000000
2007-02-05	3.293741	2.484370	2.414889	99.181993	2.424386	892692	0	2.480903	2.255598	2.368250	1.023312	0.930379	0.976846	0.386754	0	100.000000
2007-02-06	3.227201	2.439382	2.424386	99.181993	2.424386	1562964	0	2.486863	2.257136	2.371999	1.025770	0.931014	0.978392	0.386754	1	100.000000
...
2022-12-23	28.020000	29.067304	27.509240	1150.984796	28.134451	3636000	-1000	30.393746	24.630689	27.512218	1.080304	0.875464	0.977884	0.764416	0	301544.539073
2022-12-26	27.900000	27.846655	26.616083	1098.205897	26.844334	2280600	-1000	30.268076	24.575743	27.421909	1.127541	0.915491	1.021516	0.227958	0	301544.539073
2022-12-27	27.190001	27.122205	26.298516	1086.432195	26.556540	3627100	-1000	30.075381	24.525301	27.300341	1.132504	0.923513	1.028008	0.173603	0	301544.539073
2022-12-28	26.879999	27.836731	26.566464	1132.715180	27.687872	2841600	-1000	29.484440	24.822493	27.153466	1.064886	0.896511	0.980699	0.712526	0	301544.539073
2022-12-29	28.000000	28.203918	27.112282	1121.347468	27.410002	2456000	-1000	29.199284	24.927032	27.063158	1.065278	0.909414	0.987346	0.539651	0	301544.539073
3947 rows × 16 columns																

3947 rows x 16 columns