Nome: Adriano da Silva de Carvalho

Nº USP: 13692400

Alocação Sistemática

Código utilizando o conceito de **Random Forests**

```
from sklearn import tree
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier

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

sns.set(style='whitegrid')

# Carregar dados
dados = pd.read_excel('Dados_Classes_Indices.xlsx', engine='openpyxl')
dados.set_index(keys='Data', inplace=True)

dados_chg = dados.pct_change()
dados_chg.fillna(0, inplace=True)

# Cálculo do Momentum de 1, 3 e 6 meses para todos os índices
def calcular_momentum(dados, periodo):
    dados_mom = dados.copy()
    dados_mom.iloc[0:periodo] = 0
    for ind in range(periodo, len(dados.index)):
        dados_mom.iloc[ind] = dados.iloc[ind] / dados.iloc[ind - periodo]
    return dados_mom

dados_mom1 = calcular_momentum(dados, 1)
dados_mom3 = calcular_momentum(dados, 3)
dados_mom6 = calcular_momentum(dados, 6)

# Criando o data frame com informações para o algoritmo de aprendizado
ativos = ['IBOV', 'SELIC-ACC', 'IMAB', 'SP500BR']
dados_apr = dados_chg[ativos].copy()

# Selecionando o Momentum dos ativos para entrada do algoritmo
for ativo in ativos:
    dados_apr[f'{ativo}_MOM1'] = dados_mom1[ativo]
    dados_apr[f'{ativo}_MOM3'] = dados_mom3[ativo]
    dados_apr[f'{ativo}_MOM6'] = dados_mom6[ativo]

# Criando as colunas com os resultados de alocação ideais (saída do algoritmo)
dados_apr['BEST-BUY'] = np.argmax(dados_apr[ativos].reset_index().drop(['Data'], axis=1).to_numpy(), axis=1)

# Criando os vetores Numpy com as entradas (din) e saídas desejadas (dout)
entradas = [f'{ativo}_MOM1' for ativo in ativos] + [f'{ativo}_MOM3' for ativo in ativos] + [f'{ativo}_MOM6' for ativo in ativos]
din = dados_apr[entradas].reset_index().drop(['Data'], axis=1).to_numpy()
dout = dados_apr[['BEST-BUY']].reset_index().drop(['Data'], axis=1).to_numpy()

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

# Número de samples para treinamento
n_train = 100

# Separando os dados em conjunto de treinamento e validação
train_in = din[12:12 + n_train]
train_out = dout[13:13 + n_train]

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

# Treinamento com Árvores de decisão ou Random Forests
clf = RandomForestClassifier(random_state=1, max_depth=20)

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

# Avaliando os resultados
```

```
# Avaliando os resultados
y_pred = clf.predict(train_in)
print("Accuracy train:", metrics.accuracy_score(train_out, y_pred))

y_pred = clf.predict(val_in)
print("Accuracy validation:", metrics.accuracy_score(val_out, y_pred))

y_pred = clf.predict(din)

# Copiando as saídas do algoritmo de aprendizado para o Data frame
dados_apr['BEST-BUY-APR'] = y_pred
dados_apr['BEST-BUY-APR'] = dados_apr['BEST-BUY-APR'].shift(1)

# Calculando o resultado acumulado do investimento utilizando aprendizado
for ativo in ativos:
    dados_apr[f'{ativo}-CHG'] = dados_apr[ativo] * (dados_apr['BEST-BUY-APR'] == ativos.index(ativo))

dados['APR-ACC'] = (1 + dados_apr[[f'{ativo}-CHG' for ativo in ativos]].sum(axis=1)).cumprod()

# Gráfico de comparação IBOV x SELIC x Aprendizado
dados = dados * 100 / dados.iloc[n_train]
dados[['IBOV', 'SELIC-ACC', 'APR-ACC']].iloc[n_train:].plot(figsize=(15, 5))

# Retorno e volatilidade IBOV x SELIC x Aprendizado
ref_data = n_train
periodo = int(len(dados.index[ref_data + 1:]) / 12)
print("Periodo:", dados.index[ref_data + 1], "-", dados.index[-1], '(', periodo, ')')

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

```
Data samples: 248
Accuracy train: 1.0
Accuracy validation: 0.31851851851851853
Periodo: Fev-2012 - Abr-2024 ( 12 )
Retorno acumulado:
 IBOV         1.996515
SELIC-ACC    2.916759
APR-ACC      4.819669
dtype: float64
Retorno anualizado:
 IBOV         0.059309
SELIC-ACC    0.093306
APR-ACC      0.140035
dtype: float64
Vol anualizada:
 IBOV         0.219387
SELIC-ACC    0.010207
APR-ACC      0.165381
dtype: float64
```

Código utilizando o conceito de **Multi Layer Perceptron**

```
from sklearn.neural_network import MLPClassifier
```

```
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')

# Carregar dados
dados = pd.read_excel('Dados_Classes_Indices.xlsx', engine='openpyxl')
dados.set_index(keys='Data', inplace=True)

dados_chg = dados.pct_change()
dados_chg.fillna(0, inplace=True)

# Função para calcular o momentum
def calcular_momentum(dados, periodo):
    dados_mom = dados.copy()
    dados_mom.iloc[0:periodo] = 0
    for ind in range(periodo, len(dados.index)):
        dados_mom.iloc[ind] = dados.iloc[ind] / dados.iloc[ind - periodo]
    return dados_mom

# Calcular momentum para períodos de 1, 3 e 6 meses
dados_mom1 = calcular_momentum(dados, 1)
dados_mom3 = calcular_momentum(dados, 3)
dados_mom6 = calcular_momentum(dados, 6)

# Selecionar ativos
ativos = ['IBOV', 'SELIC-ACC', 'IMAB', 'SP500BR']
dados_apr = dados_chg[ativos].copy()

# Adicionar colunas de momentum
for ativo in ativos:
    dados_apr[f'{ativo}_MOM1'] = dados_mom1[ativo]
    dados_apr[f'{ativo}_MOM3'] = dados_mom3[ativo]
    dados_apr[f'{ativo}_MOM6'] = dados_mom6[ativo]

# Criar coluna de melhor compra (rótulo para aprendizado supervisionado)
dados_apr['BEST-BUY'] = np.argmax(dados_apr[ativos].reset_index().drop(['Data'], axis=1).to_numpy(), axis=1)

# Criar vetores Numpy para entrada e saída
entradas = [f'{ativo}_MOM1' for ativo in ativos] + [f'{ativo}_MOM3' for ativo in ativos] + [f'{ativo}_MOM6' for ativo in ativos]
din = dados_apr[entradas].reset_index().drop(['Data'], axis=1).to_numpy()
dout = dados_apr[['BEST-BUY']].reset_index().drop(['Data'], axis=1).to_numpy()

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

# Número de samples para treinamento
n_train = 100

# Separar dados em treinamento e validação
train_in = din[12:12 + n_train]
train_out = dout[13:13 + n_train]

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

# Treinamento com Redes Neurais MLP
clf = MLPClassifier(random_state=1, hidden_layer_sizes=(20, 2), max_iter=1000000, solver='lbfgs', activation='tanh')
clf.fit(train_in, train_out.ravel())

# Avaliação dos resultados
y_pred = clf.predict(train_in)
print("Accuracy train:", metrics.accuracy_score(train_out, y_pred))

y_pred = clf.predict(val_in)
print("Accuracy validation:", metrics.accuracy_score(val_out, y_pred))

y_pred = clf.predict(din)

# Adicionar saídas do algoritmo ao DataFrame
dados_apr['BEST-BUY-APR'] = y_pred
dados_apr['BEST-BUY-APR'] = dados_apr['BEST-BUY-APR'].shift(1)

# Calcular resultados acumulados do investimento
for ativo in ativos:
    dados_apr[f'{ativo}-CHG'] = dados_apr[ativo] * (dados_apr['BEST-BUY-APR'] == ativos.index(ativo))

dados['APR-ACC'] = (1 + dados_apr[[f'{ativo}-CHG' for ativo in ativos]].sum(axis=1)).cumprod()

# Gráfico de comparação
```

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

# Retorno e volatilidade
ref_data = n_train
periodo = int(len(dados.index[ref_data + 1:]) / 12)
print("Periodo:", dados.index[ref_data + 1], "-", dados.index[-1], '(', periodo, ')')

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

```
Data samples: 248
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:541: ConvergenceWarning: lbfgs failed to c
STOP: TOTAL NO. of f AND g EVALUATIONS EXCEEDS LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
Accuracy train: 0.57
Accuracy validation: 0.3333333333333333
Periodo: Fev-2012 - Abr-2024 ( 12 )
Retorno acumulado:
 IBOV          1.996515
SELIC-ACC     2.916759
APR-ACC      13.957363
dtype: float64
Retorno anualizado:
 IBOV         0.059309
SELIC-ACC    0.093306
APR-ACC      0.245662
dtype: float64
Vol anualizada:
 IBOV         0.219387
SELIC-ACC    0.010207
APR-ACC      0.154954
dtype: float64
```

Conclusão da utilização de **Alocação Sistemática**

Com a adição de 2 novos ativos "IMAB" e "SP500BR", vamos que ao utilizar a biblioteca de redes neurais de Multi Layer Perception foi obtido um resultado muito expressivo comparado com os vistos em aula.