@@ -1,174 +1,174 @@

"""Bayesian regression for latent source model and Bitcoin.

This module implements the 'Bayesian regression for latent source model' method

for predicting price variation of Bitcoin. You can read more about the method

at https://arxiv.org/pdf/1410.1231.pdf.

"""

import numpy as np

import bigfloat as bg

from numpy.linalg import norm

from sklearn import linear\_model

from sklearn.cluster import KMeans

def generate\_timeseries(prices, n):

"""Use the first time period to generate all possible time series of length n

and their corresponding label.

Args:

prices: A numpy array of floats representing prices over the first time

period.

n: An integer (180, 360, or 720) representing the length of time series.

Returns:

A 2-dimensional numpy array of size (len(prices)-n) x (n+1). Each row

represents a time series of length n and its corresponding label

(n+1-th column).

"""

m = len(prices) - n

ts = np.empty((m, n + 1))

for i in range(m):

ts[i, :n] = prices[i:i + n]

ts[i, n] = prices[i + n] - prices[i + n - 1]

return ts

def find\_cluster\_centers(timeseries, k):

"""Cluster timeseries in k clusters using k-means and return k cluster centers.

Args:

timeseries: A 2-dimensional numpy array generated by generate\_timeseries().

k: An integer representing the number of centers (e.g. 100).

Returns:

A 2-dimensional numpy array of size k x num\_columns(timeseries). Each

row represents a cluster center.

"""

k\_means = KMeans(n\_clusters=k)

k\_means.fit(timeseries)

return k\_means.cluster\_centers\_

def choose\_effective\_centers(centers, n):

"""Choose n most effective cluster centers with high price variation."""

return centers[np.argsort(np.ptp(centers, axis=1))[-n:]]

def predict\_dpi(x, s):

"""Predict the average price change Δp\_i, 1 <= i <= 3.

Args:

x: A numpy array of floats representing previous 180, 360, or 720 prices.

s: A 2-dimensional numpy array generated by choose\_effective\_centers().

Returns:

A big float representing average price change Δp\_i.

"""

num = 0

den = 0

for i in range(len(s)):

y\_i = s[i, len(x)]

x\_i = s[i, :len(x)]

exp = bg.exp(-0.25 \* norm(x - x\_i) \*\* 2)

num += y\_i \* exp

den += exp

return num / den

def linear\_regression\_vars(prices, v\_bid, v\_ask, s1, s2, s3):

"""Use the second time period to generate the independent and dependent variables

in the linear regression model Δp = w0 + w1 \* Δp1 + w2 \* Δp2 + w3 \* Δp3 + w4 \* r.

Args:

prices: A numpy array of floats representing prices over the second time

period.

v\_bid: A numpy array of floats representing total volumes people are

willing to buy over the second time period.

v\_ask: A numpy array of floats representing total volumes people are

willing to sell over the second time period.

s1: A 2-dimensional numpy array generated by choose\_effective\_centers()

s2: A 2-dimensional numpy array generated by choose\_effective\_centers().

s3: A 2-dimensional numpy array generated by choose\_effective\_centers().

Returns:

A tuple (X, Y) representing the independent and dependent variables in

the linear regression model. X is a 2-dimensional numpy array and each

row represents [Δp1, Δp2, Δp3, r]. Y is a numpy array of floats and

each array element represents Δp.

"""

X = np.empty((len(prices) - 721, 4))

Y = np.empty(len(prices) - 721)

for i in range(720, len(prices) - 1):

dp = prices[i + 1] - prices[i]

dp1 = predict\_dpi(prices[i - 180:i], s1)

dp2 = predict\_dpi(prices[i - 360:i], s2)

dp3 = predict\_dpi(prices[i - 720:i], s3)

r = (v\_bid[i] - v\_ask[i]) / (v\_bid[i] + v\_ask[i])

X[i - 720, :] = [dp1, dp2, dp3, r]

Y[i - 720] = dp

return X, Y

def find\_parameters\_w(X, Y):

"""Find the parameter values w for the model which best fits X and Y.

Args:

X: A 2-dimensional numpy array representing the independent variables

in the linear regression model.

Y: A numpy array of floats representing the dependent variables in the

linear regression model.

Returns:

A tuple (w0, w1, w2, w3, w4) representing the parameter values w.

"""

clf = linear\_model.LinearRegression()

clf.fit(X, Y)

w0 = clf.intercept\_

w1, w2, w3, w4 = clf.coef\_

return w0, w1, w2, w3, w4

def predict\_dps(prices, v\_bid, v\_ask, s1, s2, s3, w):

"""Predict average price changes (final estimations Δp) over the third

time period.

Args:

prices: A numpy array of floats representing prices over the third time

period.

v\_bid: A numpy array of floats representing total volumes people are

willing to buy over the third time period.

v\_ask: A numpy array of floats representing total volumes people are

willing to sell over the third time period.

s1: A 2-dimensional numpy array generated by choose\_effective\_centers()

s2: A 2-dimensional numpy array generated by choose\_effective\_centers().

s3: A 2-dimensional numpy array generated by choose\_effective\_centers().

w: A tuple (w0, w1, w2, w3, w4) generated by find\_parameters\_w().

Returns:

A numpy array of floats. Each array element represents the final

estimation Δp.

"""

dps = []

w0, w1, w2, w3, w4 = w

for i in range(720, len(prices) - 1):

dp1 = predict\_dpi(prices[i - 180:i], s1)

dp2 = predict\_dpi(prices[i - 360:i], s2)

dp3 = predict\_dpi(prices[i - 720:i], s3)

r = (v\_bid[i] - v\_ask[i]) / (v\_bid[i] + v\_ask[i])

dp = w0 + w1 \* dp1 + w2 \* dp2 + w3 \* dp3 + w4 \* r

dps.append(float(dp))

return dps

def evaluate\_performance(prices, dps, t, step):

"""Use the third time period to evaluate the performance of the algorithm.

Args:

prices: A numpy array of floats representing prices over the third time

period.

dps: A numpy array of floats generated by predict\_dps().

t: A number representing a threshold.

step: An integer representing time steps (when we make trading decisions).

dps: A numpy array of floats generated by predict\_dps().

Returns:

A number representing the bank balance.

"""

bank\_balance = 0

position = 0

for i in range(720, len(prices) - 1, step):

# long position - BUY

if dps[i - 720] > t and position <= 0:

position += 1

bank\_balance -= prices[i]

# short position - SELL

if dps[i - 720] < -t and position >= 0:

position -= 1

bank\_balance += prices[i]

# sell what you bought

if position == 1:

bank\_balance += prices[len(prices) - 1]

# pay back what you borrowed

if position == -1:

bank\_balance -= prices[len(prices) - 1]

return bank\_balance