alchemist$_{l}ibDocumentation$

Release 0.1

Carniel Giorgio

Jun 07, 2018

Contents


ALCHEMIST

CHAPTER 1

Description

Alchemist_lib is an automatic trading library for cryptocurrencies that allow to personalize the portfolio based on a specific strategy.

CHAPTER 2

Features

- Easy to use: The interface is similar to zipline, a popular backtesting software for stocks.
- Portfolio personalization: You can choose the weight of every element on the portfolio.
- Most common technical analysis indicators already integrated.
- Execute orders on the most famous exchanges.
- Possibility to visualize the asset allocation and the portfolio value charts for every strategy thanks to alchemist-view.
- Fully documented and hosted on readthedocs.

CHAPTER 3

Supported Exchanges

The following exchanges are available to trade on:

- Poloniex
- Bittrex

CHAPTER 4

Requirements

- Python3
- Mysql

CHAPTER 5

Installation

See the installing documentation.

CHAPTER 6

Code example

Strategy description: *Hold a portfolio equally composed by Ethereum and BitcoinCash.*

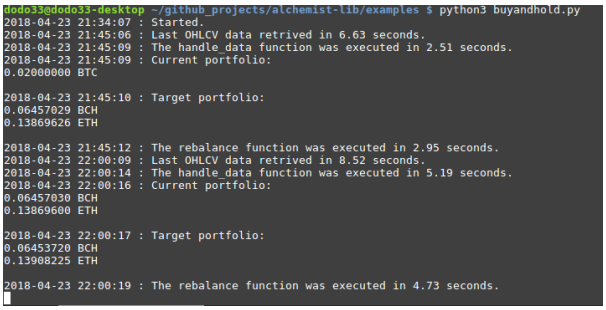
```
from alchemist_lib.portfolio import LongsOnlyPortfolio
from alchemist_lib.broker import PoloniexBroker
from alchemist_lib.tradingsystem import TradingSystem
import alchemist_lib.exchange as exch
import pandas as pd


def set_weights(df):
    df["weight"] = 0.5 #Because there are just two assets.
    return df

def select_universe(session):
    poloniex_assets = exch.get_assets(session = session, exchange_name = "poloniex")

    my_universe = []
    for asset in poloniex_assets:
        if asset.ticker == "ETH" or asset.ticker == "BCH":
            my_universe.append(asset)
    return my_universe

def handle_data(session, universe):
    #The value of alpha is useless in this case.
    df = pd.DataFrame(data = {"asset" : universe, "alpha" : 0}, columns = ["asset",
↪"alpha"]).set_index("asset")
    return df


algo = TradingSystem(name = "BuyAndHold",
                     portfolio = LongsOnlyPortfolio(capital = 0.02),
                     set_weights = set_weights,
                     select_universe = select_universe,
                     handle_data = handle_data,
```

(continues on next page)

(continued from previous page)

```
                    broker = PoloniexBroker(api_key = "APIKEY",
                                            secret_key = "SECRETKEY"),
                    paper_trading = True)
algo.run(delay = "15M", frequency = 1)
```

6.1 Screenshot



CHAPTER 7

Basic concepts

Alchemist_lib works with three methods:

- set_weights
- select_universe
- handle_data

set_weights is used to set the weight that an asset has respect the others within the portfolio. The sum of every weight must be close to 1. Must returns a pandas dataframe with two columns: “asset” and “alpha”, where “asset” is the index.

select_universe filters the assets saved on the database and returns just the ones the strategy will take into consideration.

handle_data is the most importat one because it manages the trading logic. Must returns a pandas dataframe with two columns: “asset” and “alpha”, where “asset” is the index.

You can find other examples in the `examples` directory.

CHAPTER 8

Reporting bugs

A bug tracker is provided by Github.

8.1 Contents

8.1.1 Install

Installing requirements

Lets install python3 and mysql.

GNU/Linux

First of all to make sure that everything is up to date, let's update and upgrade the system with apt-get.

```
$ sudo apt-get update
$ sudo apt-get -y upgrade
```

Probably python3 is already installed so let's check.

```
$ python3 -V
```

If the command above returns something like `Python 3.5.2` it's all ok. Otherwise install python with the following command.

```
$ sudo apt-get install python3
```

To manage software packages for Python, let's install pip.

```
$ sudo apt-get install python3-pip
```

A more detailed guide can be found on Digital Ocean.

Installing MySQL can be done by runnung the following command.

```
$ sudo apt-get install mysql-server
```

The MySQL/Python connector can be installed with:

```
$ sudo apt-get install python3-mysql.connector
```

Installing alchemist_lib

Installing with `pip`:

If python3-pip is already installed:

```
$ sudo pip3 install alchemist_lib
$ sudo pip3 install git+https://github.com/femtotrader/pandas_talib.git
$ sudo pip3 install https://github.com/s4w3d0ff/python-poloniex/archive/v0.4.7.zip
```

If you don't have pip installed, you can easily install it by downloading and running get-pip.py.

Cloning the repository with `git`:

If git is already installed:

```
$ git clone https://github.com/Dodo33/alchemist-lib
$ cd alchemist-lib
$ python3 setup.py install

$ sudo pip3 install git+https://github.com/femtotrader/pandas_talib.git
$ sudo pip3 install https://github.com/s4w3d0ff/python-poloniex/archive/v0.4.7.zip
```

Important

After the installation it's important to specify mysql credentials:

```
$ sudo alchemist populate -l "hostname" -u "username" -p "password" -d "database_name"
```

8.1.2 Beginner Tutorial

Basics

Alchemist_lib works with three methods:

- set_weights
- select_universe
- handle_data

set_weights is used to set the weight that an asset has respect the others within the portfolio. The sum of every weight must be close to 1. The `df` parameter is the dataframe returned by `handle_data`. Must returns a pandas dataframe with two columns: "asset" and "weight", where "asset" is the index.

select_universe have to returns a list of assets the strategy will take into consideration. If you want all the assets traded on a specific exchange just call the `get_assets` function of `alchemist_lib.exchange`.

handle_data is the most importat one because it manages the trading logic. The `universe` parameter is the list returned by `select_universe`. Must returns a pandas dataframe with two columns: "asset" and "alpha", where "asset" is the index.

To start the strategy you just need to instantiate the `TradingSystem` class and call the `run` method.

Note: Remember to test the strategy with real-time data before going live, it can be done setting `paper_trading = True`.

First strategy

Lets take a look at a very simple strategy from the `examples` directory, `buyandhold.py`.

Strategy description: *Hold a portfolio equally composed by Ethereum and BitcoinCash.*

First of all we must import all the things we need.

```
from alchemist_lib.portfolio import LongsOnlyPortfolio
from alchemist_lib.broker import PoloniexBroker
from alchemist_lib.tradingsystem import TradingSystem
import alchemist_lib.exchange as exch
import pandas as pd
```

Then we select which assets we want to buy and hold. Just ETH and BCH in this example:

```
def select_universe(session):
    poloniex_assets = exch.get_assets(session = session, exchange_name = "poloniex")

    my_universe = []
    for asset in poloniex_assets:
        if asset.ticker == "ETH" or asset.ticker == "BCH":
            my_universe.append(asset)
    return my_universe
```

In this case the `handle_data` method is useless so lets set a random value for the "alpha" column of the dataframe.

```
def handle_data(session, universe):
    df = pd.DataFrame(data = {"asset" : universe, "alpha" : 0}, columns = ["asset",
↪"alpha"]).set_index("asset")
    return df
```

We want to hold two assets (ETH and BCH) so every one must be 50% of the portfolio value.

```
def set_weights(df):
    df["weight"] = 0.5
    return df
```

Make it starts in paper trading mode, every 4 hours.

```
algo = TradingSystem(name = "BuyAndHold",
                     portfolio = LongsOnlyPortfolio(capital = 0.01),
```

(continues on next page)

(continued from previous page)

```
                    set_weights = set_weights,
                    select_universe = select_universe,
                    handle_data = handle_data,
                    broker = PoloniexBroker(api_key = "APIKEY",
                                            secret_key = "SECRETKEY"),
                    paper_trading = True)
algo.run(delay = "4H", frequency = 1)
```

Execution

Just type:

```
$ python3 buyandhold.py
```

A log file called `buyandhold.log` will be created.

Example

Another example, a little bit more complex is `emacrossover.py`.

Strategy description: *Hold a portfolio composed by top 5 assets by volume whose EMA 10 is above the EMA 21. Rebalance it every hour.*

Code:

```
from alchemist_lib.portfolio import LongsOnlyPortfolio
from alchemist_lib.broker import BittrexBroker
from alchemist_lib.tradingsystem import TradingSystem
from alchemist_lib.factor import Factor
import pandas as pd
import alchemist_lib.exchange as exch


def set_weights(df):
    alphas_sum = df["alpha"].sum()
    for asset, alpha in zip(df.index.values, df["alpha"]):
        df.loc[asset, "weight"] = alpha / alphas_sum

    return df

def select_universe(session):
    return exch.get_assets(session = session, exchange_name = "bittrex")

def handle_data(session, universe):
    fct = Factor(session = session)
    prices = fct.history(universe = universe, field = "close", timeframe = "1H",␣
↪window_length = 21)

    ema10 = fct.ExponentialMovingAverage(values = prices, window_length = 10, field =
↪"close").rename(columns = {"ExponentialMovingAverage" : "ema10"})
    ema21 = fct.ExponentialMovingAverage(values = prices, window_length = 21, field =
↪"close").rename(columns = {"ExponentialMovingAverage" : "ema21"})

    concated = pd.concat([ema10, ema21], axis = 1)
    concated = concated.loc[concated["ma10"] > concated["ma21"], :]
```

(continues on next page)

(continued from previous page)

```
    vol = fct.history(universe = concated.index.values, field = "volume", timeframe =
↪"1H", window_length = 1)

    df = pd.concat([concated, vol], axis = 1)
    df = df[["volume"]].rename(columns = {"volume" : "alpha"})

    if len(df) > 5:
        df = df.sort_values(by = "volume", ascending = False)
        df = df.head(5)

    return df


algo = TradingSystem(name = "MovingAverageCrossover",
                     portfolio = LongsOnlyPortfolio(capital = 0.1),
                     set_weights = set_weights,
                     select_universe = select_universe,
                     handle_data = handle_data,
                     broker = BittrexBroker(api_key = "APIKEY",
                                            secret_key = "SECRETKEY"),
                     paper_trading = True)
algo.run(delay = "1H", frequency = 1)
```

To execute it:

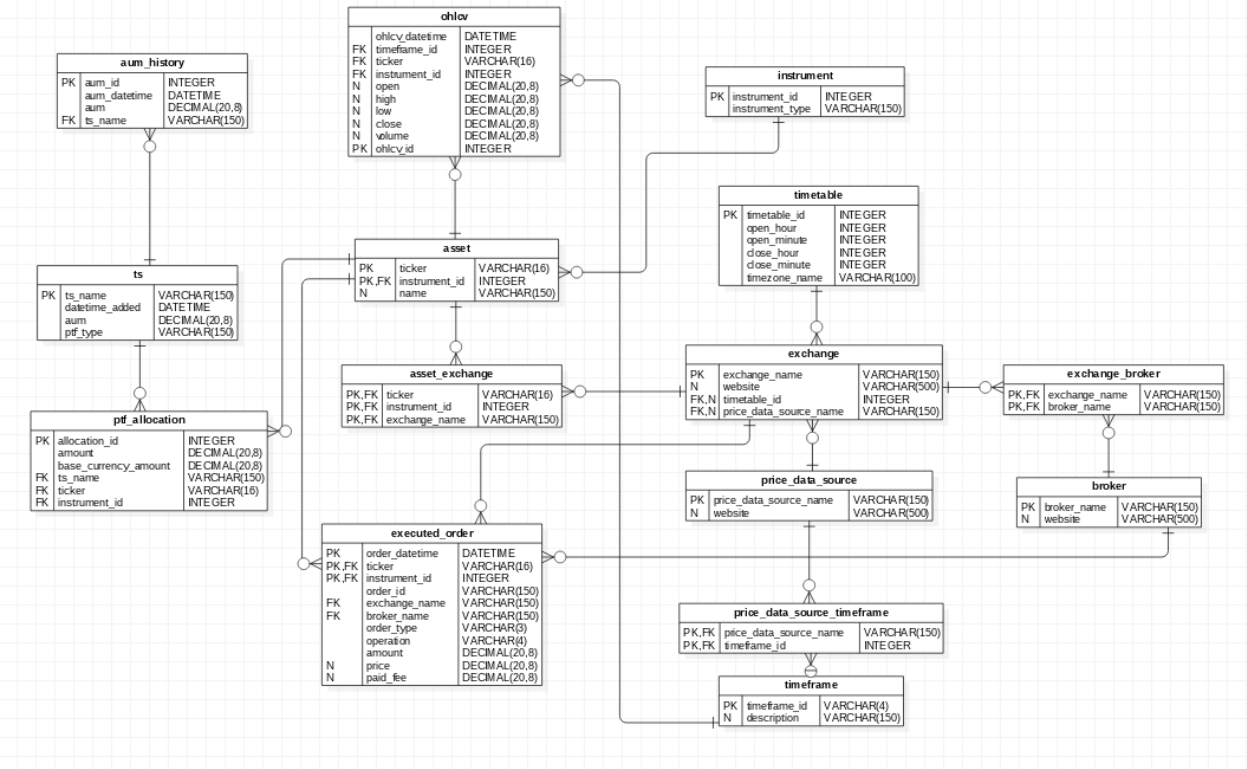
```
$ python3 emacrossover.py
```

Conclusion

These were some basic examples of how alchemist_lib works. Take a look at the `example` folder for more examples.

8.1.3 Database

UML Diagram



Tables

Asset

Instrument

Broker

Exchange

PriceDataSource

TradingSystem

Timeframe

Timetable

AumHistory

PtfAllocation

Ohlcv

ExecutedOrder

8.1.4 API Reference

Trading system

Factor

Factor autoclass

Datafeed

__init__

ohlcv

poloniexdatafeed

bittrexdatafeed

Broker

broker

poloniexbroker

bittrexbroker

Portfolio

portfolio

longsonly

Exchange

exchange

__init__

poloniexexchange

bittrexexchange

Populate

saver

populate

__init__

poloniexpopulate

bittrexpopulate

8.1.5 License

MIT License