TradingBot Documentation

Release 1.0.0

Alberto Cardellini

Dec 14, 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION

TradingBot is an autonomous trading system that uses customised strategies to trade in the London Stock Exchange market. This documentation provides an overview of the system, explaining how to create new trading strategies and how to integrate them with TradingBot. Explore the next sections for a detailed documentation of each module too.

1.1 System Overview

TradingBot is a python program with the goal to automate the trading of stocks in the London Stock Exchange market. It is designed around the idea that to trade in the stock market you need a **strategy**: a strategy is a set of rules that define the conditions where to buy, sell or hold a certain market. TradingBot design lets the user implement a custom strategy without the trouble of developing all the boring stuff to make it work.

The following sections give an overview of the main components that compose TradingBot.

1.1.1 TradingBot

TradingBot is the main entiy used to initialised all the components that will be used during the main routine. It reads the configuration file and the credentials file, it creates the configured strategy instance, the broker interface and it handle the processing of the markets with the active strategy.

1.1.2 Broker interface

TradingBot requires an interface with an executive broker in order to open and close trades in the market. The broker interface is initialised in the `TradingBot` module and it should be independent from its underlying implementation.

At the current status, the only supported broker is IGIndex. This broker provides a very good set of API to analyse the market and manage the account. TradingBot makes also use of other 3rd party services to fetch market data such as price snapshot or technical indicators.

1.1.3 Strategy

The `Strategy` is the core of the TradingBot system. It is a generic template class that can be extended with custom functions to execute trades according to the personalised strategy.

How to use your own strategy

Anyone can create a new strategy from scratch in a few simple steps. With your own strategy you can define your own set of rules to decide whether to buy, sell or hold a specific market.

1. Setup your development environment (see `README.md`)
2. Create a new python module inside the Strategy folder :

```
cd Strategies
touch my_strategy.py
```

3. Edit the file and add a basic strategy template like the following:

```
import os
import inspect
import sys
import logging

# Required for correct import path
currentdir = os.path.dirname(os.path.abspath(inspect.getfile(inspect.
↪currentframe())))
parentdir = os.path.dirname(currentdir)
sys.path.insert(0,parentdir)

from Components.Utils import Utils, Interval, TradeDirection
from .Strategy import Strategy
# Import any other required module

class my_strategy(Strategy): # Extends Strategy module
    """
    Description of the strategy
    """
    def read_configuration(self, config):
        # Read from the config json and store config parameters
        pass

    def initialise(self):
        # Initialise the strategy
        pass

    def fetch_datapoints(self, market):
        """
        Fetch any required datapoints (historic prices, indicators, etc.).
        The object returned by this function is passed to the 'find_trade_signal()'
        function 'datapoints' argument
        """
        # As an example, this means the strategy needs 50 data point of
        # of past prices from the 1-hour chart of the market
        return self.broker.get_prices(market.epic, Interval.HOUR, 50)

    def find_trade_signal(self, market, prices):
        # Here is where you want to implement your own code!
        # The market instance provide information of the market to analyse while
        # the prices dictionary contains the required price datapoints
```

(continues on next page)

(continued from previous page)

```
        # Returns the trade direction, stop level and limit level
        # As an examle:
        return TradeDirection.BUY, 90, 150

    def backtest(self, market, start_date, end_date):
        # This is still a work in progress
        # The idea here is to perform a backtest of the strategy for the given␣
↪market
        return {"balance": balance, "trades": trades}
```

4. Add the implementation for these functions:
    - *read_configuration*: `config` is the configuration wrapper instance loaded from the configuration file
    - *initialise*: initialise the strategy or any internal members
    - *fetch_datapoints*: fetch the required past price datapoints
    - *find_trade_signal*: it is the core of your custom strategy, here you can use the broker interface to decide if trade the given epic
    - *backtest*: backtest the strategy for a market within the date range
5. `Strategy` parent class provides a `Broker` type internal member that can be accessed with `self.broker`. This member is the TradingBot broker interface and provide functions to fetch market data, historic prices and technical indicators. See the *Modules* section for more details.
6. `Strategy` parent class provides access to another internal member that list the current open position for the configured account. Access it with `self.positions`.
7. Edit the `StrategyFactory` module inporting the new strategy and adding its name to the `StrategyNames` enum. Then add it to the *make* function
8. Edit the `TradingBot` configuration file adding a new section for your strategy parameters
9. Create a unit test for your strategy
10. Share your strategy creating a Pull Request :)

1.2 Modules

TradingBot is composed by different modules organised by their nature. Each section of this document provide a description of the module meaning along with the documentation of its internal members.

1.2.1 TradingBot

1.2.2 Components

The `Components` module contains the components that provides services used by TradingBot.

MarketProvider

Enums

TimeProvider

Enums

Backtester

Configuration

Utils

Enums

Exceptions

1.2.3 Broker

The `Broker` class is the wrapper of all the trading services and provides the main interface for the `strategies` to access market data and perform trades.

AbstractInterfaces

IGInterface

Enums

AVInterface

Enums

YFinanceInterface

Broker

BrokerFactory

1.2.4 Interfaces

The `Interfaces` module contains all the interfaces used to exchange information between different TradingBot components. The purpose of this module is have clear internal API and avoid integration errors.

Market

MarketHistory

MarketMACD

Position

1.2.5 Strategies

The `Strategies` module contains the strategies used by TradingBot to analyse the markets. The `Strategy` class is the parent from where any custom strategy **must** inherit from. The other modules described here are strategies available in TradingBot.

Strategy

StrategyFactory

SimpleMACD

Weighted Average Peak Detection

Simple Bollinger Bands