TradingBot Documentation

Release 1.0.0

Alberto Cardellini

Aug 31, 2019

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 *TradingBot*)
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 Interfaces.Broker import Interval
from .Strategy import Strategy
from Utility.Utils import Utils, TradeDirection
# 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 get_price_settings(self):
        """
        Returns the price settings required by the strategy
        """
        # As an example, this means the strategy needs 50 data point of
        # of past prices from the 1-hour chart of the market
        # Return a list of tuple
        return [(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
        # Returns the trade direction, stop level and limit level
        # As an examle:
        return TradeDirection.BUY, 90, 150

    def get_seconds_to_next_spin(self):
        # Return the amount of seconds between each spin of the strategy
        # Each spin analyses all the markets in a list/watchlist
        # Some strategies might require to run once a day, while other might
        # need to run continuosly, here you can make your decision
```

4. Add the implementation for these functions:
    - *read_configuration*: `config` is the json object loaded from the `config.json` file
    - *initialise*: initialise the strategy or any internal members
    - *get_price_settings*: define 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
    - *get_seconds_to_next_spin*: the *find_trade_signal* is called for every epic requested. After that TradingBot will wait for the amount of seconds defined in this function
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

```
  def make_strategy(self, strategy_name):
      if strategy_name == StrategyNames.SIMPLE_MACD.value:
          return SimpleMACD(self.config, self.broker)
      elif strategy_name == StrategyNames.FAIG.value:
          return FAIG_iqr(self.config, self.broker)
      elif strategy.name == StrateyNames.MY_STRATEGY.value:
          return MY_STRATEGY(self.config, self.broker)
      else:
          logging.error('Impossible to create strategy {}. It does not exist'.
↪format(strategy_name))
```

8. Edit the `config.json` 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 Interfaces

The `Interfaces` module contains all those interfaces with external services used by TradingBot. 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.

IGInterface

AVInterface

Enums

Broker

Enums

Market

class Interfaces.Market.**Market**
Represent a tradable market with latest price information

MarketProvider

class Interfaces.MarketProvider.**MarketProvider**(*config*, *broker*)
Provide markets from different sources based on configuration. Supports market lists, dynamic market exploration or watchlists

get_market_from_epic(*epic*)
Given a market epic id returns the related market snapshot

next()
Return the next market from the configured source

reset()
Reset internal market pointer to the beginning

Enums

class Interfaces.MarketProvider.**MarketSource**
Available market sources: local file list, watch list, market navigation through API, etc.

1.2.3 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

1.2.4 Utils

Common utility classes and methods

Utils

Enums

Exceptions

1.3 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

1.3.1 [1.1.0] - 2019-09-01

Changed

- Replaced bash script with python
- Moved sources in `src` installation folder
- Corrected IGInterface numpy dependency
- Added Pipenv integration
- Exported logic from Custom Strategy to simplify workflow
- Added dev-requirements.txt for retro compatibility
- Updated Sphinx documentation

1.3.2 [1.0.1] - 2019-05-09

Changed

- Updated renovate configuration

1.3.3 [1.0.0] - 2019-04-21

Added

- Initial release

CHAPTER

TWO

TRADINGBOT

This is an attempt to create an autonomous market trading script using the IG REST API and any other available data source for market prices.

TradingBot is meant to be a "forever running" process that keeps analysing the markets and taking actions whether the conditions are met. It is halfway from an academic project and a real useful piece of software, I guess I will see how it goes :)

The main goal of this project is to provide the capability to write a custom trading strategy with the minimum effort. TradingBot handle all the boring stuff.

All the credits for the FAIG_iqr strategy goes to GitHub user @tg12 who is the creator of the first script version and gave me a good starting point for this project. Thank you.

CHAPTER

THREE

DEPENDENCIES

- Python 3.5+
- Pipenv

View file `Pipfile` for the full list of required python packages.

CHAPTER

FOUR

INSTALL

First if you have not yet done so, install python 3.5+ and pipenv

```
sudo apt-get update && sudo apt-get install python3 python3-pip
sudo -H pip3 install -U pipenv
```

Clone this repo in your workspace and setup the python virtual environment by running the following commands in the repository root folder

```
pipenv install --three
```

You can install development packages adding the flag `--dev`

The following step is to install TradingBot:

```
sudo ./install.py
```

All necessary files are copied in `/opt/TradingBot` by default. It is recommended to add this path to your `PATH` environment variable.

The last step is to set file permissions for your user on the installed folders with the following command:

```
sudo chown -R $USER: $HOME/.TradingBot
```

CHAPTER

FIVE

SETUP

Login to your IG Dashboard

- Obtain an API KEY from the settings panel
- If using the demo account, create demo credentials
- Take note of your spread betting account ID (demo or real)
- Visit AlphaVantage website: `https://www.alphavantage.co`
- Request a free api key
- Insert these info in a file called `.credentials`

This must be in json format

```
{
    "username": "username",
    "password": "password",
    "api_key": "apikey",
    "account_id": "accountId",
    "av_api_key": "apiKey"
}
```

- Copy the `.credentials` file into the `$HOME/.TradingBot/data` folder
- Revoke permissions to read the file .. code-block:: guess

    cd data sudo chmod 600 $HOME/.TradingBot/data/.credentials

5.1 Market source

There are different ways to define which markets to analyse with TradinbgBot. You can select your preferred option in the `config.json` file with the `market_source` parameter:

- **Local file**

You can create a file `epic_ids.txt` containg IG epics of the companies you want to monitor. You need to copy this file into the `data` folder.

- **Watchlist**

You can use an IG watchlist, TradingBot will analyse every market added to the selected watchlist

- **API**

TradingBot navigates the IG markets dynamically using the available API call to fetch epic ids.

5.2 Configuration file

The `config.json` file is in the `config` folder and it contains several configurable parameter to personalise how TradingBot work. These are the description of each parameter:

5.2.1 General

- **max_account_usable**: The maximum percentage of account funds to use (A safe value is around 50%)
- **time_zone**: The timezone to use (i.e. 'Europe/London)
- **enable_log**: Enable the log in a file rather than on stdout
- **log_file**: Define the full file path for the log file to use, if enabled. {home} and {timestamp} placeholders are replaced with the user home directory and the timestamp when TradingBot started
- **debug_log**: Enable the debug level in the logging
- **credentials_filepath**: Filepath for the `.credentials` file
- **market_source**: The source to use to fetch the market ids. Available values as explained above are: [`list`, `watchlist`, `api`]
- **epic_ids_filepath**: The full file path for the local file containing the list of epic ids
- **watchlist_name**: The watchlist name to use as market source, if selected
- **active_strategy**: The strategy name to use. Must match one of the names in the `Strategies` section below

5.2.2 IG Interface

- **order_type**: The IG order type (MARKET, LIMIT, etc.). Do NOT change it
- **order_size**: The size of the spread bets
- **order_expiry**: The order expiry (DFB). Do NOT change it
- **order_currency**: The currency of the order (For UK shares leave it as GBP)
- **order_force_open**: Force to open the orders
- **use_g_stop**: Use guaranteed stops. Read IG terms for more info about them.
- **use_demo_account**: Trade on the DEMO IG account. If enabled remember to setup the demo account credentials too
- **controlled_risk**: Enable the controlled risk stop loss calculation. Enable only if you have a controlled risk account.
- **paper_trading**: Enable the `paper trading`. No real trades will be done on the IG account.

5.2.3 Alpha Vantage

- **enable**: Enable the use of AlphaVantage API
- **api_timeout**: Timeout in seconds between each API call

5.2.4 Strategies

Settings specific for each strategy

5.2.5 SimpleMACD

- **spin_interval**: Override the `Strategies` value
- **max_spread_perc**: Spread percentage to filter markets with high spread
- **limit_perc**: Limit percentage to take profit for each trade
- **stop_perc**: Stop percentage to stop any loss

CHAPTER

SIX

START TRADINGBOT

You can start TradingBot in your current terminal

```
/opt/TradingBot/src/TradingBot.py
```

or you can start it in detached mode, letting it run in the background

```
nohup /opt/TradingBot/src/TradingBot.py >/dev/null 2>&1 &
```

6.1 Close all the open positions

```
/opt/TradingBot/src/TradingBot.py -c
```

CHAPTER

SEVEN

STOP TRADINGBOT

To stop a TradingBot instance running in the background

```
ps -ef | grep TradingBot | xargs kill -9
```

CHAPTER

EIGHT

TEST

You can run the test from the workspace with:

```
pipenv run pytest
```

CHAPTER

NINE

DOCUMENTATION

The Sphinx documentation contains further details about each TradingBot module with source code documentation of each class member. Explanation is provided regarding how to create your own Strategy and how to integrate it with the system.

Read the documentation at:

https://tradingbot.readthedocs.io

You can build it locally with:

```
pipenv run sphinx-build -nWT -b html doc doc/_build/html
```

The generated html files will be in `doc/_build/html`.

CHAPTER

TEN

AUTOMATE

NOTE: TradingBot monitors the market opening hours and suspend the trading when the market is closed. Generally you should NOT need a cron job!

You can set up the crontab job to run and kill TradinBot at specific times. The only configuration required is to edit the crontab file adding the preferred schedule:

```
crontab -e
```

As an example this will start TradingBot at 8:00 in the morning and will stop it at 16:35 in the afternoon, every week day (Mon to Fri):

```
00 08 * * 1-5 /opt/TradingBot/src/TradingBot.py
35 16 * * 1-5 kill -9 $(ps | grep "/opt/TradingBot/src/TradingBot.py" | grep -v grep␣
↪| awk '{ print $1 }')
```

NOTE: Remember to set the correct timezone in your machine!

CHAPTER

ELEVEN

DOCKER

You can run TradingBot in a Docker container (https://docs.docker.com/). First you need to build the Docker image used by TradingBot:

```
./docker_run.sh build
```

Once the image is built you can install TradingBot and then run it in a Docker container:

```
./docker_run.sh start
```

The container will be called `dkr_trading_bot` and the logs will still be stored in the configured folder in the host machine. By default `$HOME/.TradingBot/log`.

Check the `Dockerfile` and the `docker_run.sh` for more details

To stop the TradingBot container:

```
docker kill dkr_trading_bot
```

If you need to start a bash shell into a running container

```
docker exec -it dkr_trading_bot bash
```

CHAPTER

TWELVE

CONTRIBUTING

Any contribution or suggestion is welcome, please follow the suggested workflow.

12.1 Pull Requests

To add a new feature or to resolve a bug, create a feature branch from the `develop` branch.

Commit your changes and if possible add unit/integration test cases. Eventually push your branch and create a Pull Request against `develop`.

If you instead find problems or you have ideas and suggestions for future improvements, please open an Issue. Thanks for the support!

PYTHON MODULE INDEX

i

INDEX