Hyperopt

This page explains how to tune your strategy by finding the optimal parameters, a process called hyperparameter optimization. The bot uses algorithms included in the `scikit-optimize` package to accomplish this. The search will burn all your CPU cores, make your laptop sound like a fighter jet and still take a long time.

In general, the search for best parameters starts with a few random combinations (see below for more details) and then uses Bayesian search with a ML regressor algorithm (currently ExtraTreesRegressor) to quickly find a combination of parameters in the search hyperspace that minimizes the value of the loss function.

Hyperopt requires historic data to be available, just as backtesting does (hyperopt runs backtesting many times with different parameters). To learn how to get data for the pairs and exchange you're interested in, head over to the Data Downloading section of the documentation.

Bug

Hyperopt can crash when used with only 1 CPU Core as found out in Issue #1133

Note

Since 2021.4 release you no longer have to write a separate hyperopt class, but can configure the parameters directly in the strategy. The legacy method was supported up to 2021.8 and has been removed in 2021.9.

Install hyperopt dependencies

Since Hyperopt dependencies are not needed to run the bot itself, are heavy, can not be easily built on some platforms (like Raspberry PI), they are not installed by default. Before you run Hyperopt, you need to install the corresponding dependencies, as described in this section below.

Note

Since Hyperopt is a resource intensive process, running it on a Raspberry Pi is not recommended nor supported.

Docker

The docker-image includes hyperopt dependencies, no further action needed.

Easy installation script (setup.sh) / Manual installation

```
source .venv/bin/activate
pip install -r requirements-hyperopt.txt
```

Hyperopt command reference


```
usage: freqtrade hyperopt [-h] [-v] [--logfile FILE] [-V] [-c PATH] [-d PATH]
                          [--userdir PATH] [-s NAME] [--strategy-path PATH]
                          [--recursive-strategy-search] [--freqaimodel NAME]
```

```
                          [--freqaimodel-path PATH] [-i TIMEFRAME]
                          [--timerange TIMERANGE]
                          [--data-format-ohlcv {json,jsongz,hdf5}]
                          [--max-open-trades INT]
                          [--stake-amount STAKE_AMOUNT] [--fee FLOAT]
                          [-p PAIRS [PAIRS ...]] [--hyperopt-path PATH]
                          [--eps] [--dmmp] [--enable-protections]
                          [--dry-run-wallet DRY_RUN_WALLET]
                          [--timeframe-detail TIMEFRAME_DETAIL] [-e INT]
                          [--spaces
{all,buy,sell,roi,stoploss,trailing,protection,trades,default}
[{all,buy,sell,roi,stoploss,trailing,protection,trades,default} ...]]
                          [--print-all] [--no-color] [--print-json] [-j JOBS]
                          [--random-state INT] [--min-trades INT]
                          [--hyperopt-loss NAME] [--disable-param-export]
                          [--ignore-missing-spaces] [--analyze-per-epoch]

optional arguments:
  -h, --help            show this help message and exit
  -i TIMEFRAME, --timeframe TIMEFRAME
                        Specify timeframe (`1m`, `5m`, `30m`, `1h`, `1d`).
  --timerange TIMERANGE
                        Specify what timerange of data to us
  --data-format-ohlcv {json,jsongz,hdf5}
                        Storage format for downloaded candle
                        (default: `json`).
  --max-open-trades INT
                        Override the value of the `max_open_trades`
```

```
                      configuration setting.
  --stake-amount STAKE_AMOUNT
                      Override the value of the `stake_amount` configuration
                      setting.
  --fee FLOAT         Specify fee ratio. Will be applied twice (on trade
                      entry and exit).
  -p PAIRS [PAIRS ...], --pairs PAIRS [PAIRS ...]
                      Limit command to these pairs. Pairs are space-
                      separated.
  --hyperopt-path PATH  Specify additional lookup path for Hyperopt Loss
                      functions.
  --eps, --enable-position-stacking
                      Allow buying the same pair multiple times (position
                      stacking).
  --dmmp, --disable-max-market-positions
                      Disable applying `max_open_trades` during backtest
                      (same as setting `max_open_trades` to a very high
                      number).
  --enable-protections, --enableprotections
                      Enable protections for backtesting.Will slow
                      backtesting down by a considerable amount, but will
                      include configured protections
  --dry-run-wallet DRY_RUN_WALLET, --starting-balance DRY_RUN_WALLET
                      Starting balance, used for backtesti
                      dry-runs.
  --timeframe-detail TIMEFRAME_DETAIL
                      Specify detail timeframe for backtesting (`1m`, `5m`,
                      `30m`, `1h`, `1d`).
```

```
  -e INT, --epochs INT  Specify number of epochs (default: 100).
  --spaces {all,buy,sell,roi,stoploss,trailing,protection,trades,default}
[{all,buy,sell,roi,stoploss,trailing,protection,trades,default} ...]
                        Specify which parameters to hyperopt. Space-separated
                        list.
  --print-all           Print all results, not only the best ones.
  --no-color            Disable colorization of hyperopt results. May be
                        useful if you are redirecting output to a file.
  --print-json          Print output in JSON format.
  -j JOBS, --job-workers JOBS
                        The number of concurrently running jobs for
                        hyperoptimization (hyperopt worker processes). If -1
                        (default), all CPUs are used, for -2, all CPUs but one
                        are used, etc. If 1 is given, no parallel computing
                        code is used at all.
  --random-state INT    Set random state to some positive integer for
                        reproducible hyperopt results.
  --min-trades INT      Set minimal desired number of trades for evaluations
                        in the hyperopt optimization path (default: 1).
  --hyperopt-loss NAME, --hyperoptloss NAME
                        Specify the class name of the hyperopt loss function
                        class (IHyperOptLoss). Different functions can
                        generate completely different results, since the
                        target for optimization is different.
                        Hyperopt-loss-functions are:
                        ShortTradeDurHyperOptLoss, OnlyProfitHyperOptLoss,
                        SharpeHyperOptLoss, SharpeHyperOptLossDaily,
                        SortinoHyperOptLoss, SortinoHyperOptLossDaily,
```

```
                        CalmarHyperOptLoss, MaxDrawDownHyperOptLoss,
                        MaxDrawDownRelativeHyperOptLoss,
                        ProfitDrawDownHyperOptLoss
  --disable-param-export
                        Disable automatic hyperopt parameter export.
  --ignore-missing-spaces, --ignore-unparameterized-spaces
                        Suppress errors for any requested Hyperopt spaces that
                        do not contain any parameters.
  --analyze-per-epoch   Run populate_indicators once per epoch.

Common arguments:
  -v, --verbose         Verbose mode (-vv for more, -vvv to get all messages).
  --logfile FILE        Log to the file specified. Special values are:
                        'syslog', 'journald'. See the documentation for more
                        details.
  -V, --version         show program's version number and exit
  -c PATH, --config PATH
                        Specify configuration file (default:
                        `userdir/config.json` or `config.json` whichever
                        exists). Multiple --config options may be used. Can be
                        set to `-` to read config from stdin.
  -d PATH, --datadir PATH
                        Path to directory with historical backtesting data.
  --userdir PATH, --user-data-dir PATH
                        Path to userdata directory.

Strategy arguments:
  -s NAME, --strategy NAME
```

```
                        Specify strategy class name which will be used by the
                        bot.
  --strategy-path PATH  Specify additional strategy lookup path.
  --recursive-strategy-search
                        Recursively search for a strategy in the strategies
                        folder.
  --freqaimodel NAME    Specify a custom freqaimodels.
  --freqaimodel-path PATH
                        Specify additional lookup path for freqaimodels.
```

Hyperopt checklist

Checklist on all tasks / possibilities in hyperopt

Depending on the space you want to optimize, only some of the below are required:

- define parameters with `space='buy'` - for entry signal optimization
- define parameters with `space='sell'` - for exit signal optimization

Note

`populate_indicators` needs to create all indicators any of the spaces may use, otherwise hyperopt will not work.

Rarely you may also need to create a nested class named `HyperOpt` and implement

- `roi_space` - for custom ROI optimization (if you need the ranges for the ROI parameters in the optimization hyperspace that differ from default)
- `generate_roi_table` - for custom ROI optimization (if you need the ranges for the values in the ROI table that differ from default or the number of entries (steps) in the ROI table which differs from the default 4 steps)
- `stoploss_space` - for custom stoploss optimization (if you need the range for the stoploss parameter in the optimization hyperspace that differs from default)
- `trailing_space` - for custom trailing stop optimization (if you need the ranges for the trailing stop parameters in the optimization hyperspace that differ from default)
- `max_open_trades_space` - for custom max_open_trades optimizati
  max_open_trades parameter in the optimization hyperspace that differ from default)

Quickly optimize ROI, stoploss and trailing stoploss

You can quickly optimize the spaces `roi`, `stoploss` and `trailing` without changing anything in your strategy.

```
# Have a working strategy at hand.
freqtrade hyperopt --hyperopt-loss SharpeHyperOptLossDaily --spaces roi stoploss trailing
--strategy MyWorkingStrategy --config config.json -e 100
```

Hyperopt execution logic

Hyperopt will first load your data into memory and will then run `populate_indicators()` once per Pair to generate all indicators, unless `--analyze-per-epoch` is specified.

Hyperopt will then spawn into different processes (number of processors, or `-j <n>`), and run backtesting over and over again, changing the parameters that are part of the `--spaces` defined.

For every new set of parameters, freqtrade will run first `populate_entry` `populate_exit_trend()`, and then run the regular backtesting process

After backtesting, the results are passed into the loss function, which will evaluate if this result was better or

worse than previous results.
Based on the loss function result, hyperopt will determine the next set of parameters to try in the next round of backtesting.

Configure your Guards and Triggers

There are two places you need to change in your strategy file to add a new buy hyperopt for testing:

- Define the parameters at the class level hyperopt shall be optimizing.
- Within `populate_entry_trend()` - use defined parameter values instead of raw constants.

There you have two different types of indicators: 1. `guards` and 2. `triggers`.

1. Guards are conditions like "never buy if ADX < 10", or never buy if current price is over EMA10.
2. Triggers are ones that actually trigger buy in specific moment, like "buy when EMA5 crosses over EMA10" or "buy when close price touches lower Bollinger band".

Guards and Triggers

Technically, there is no difference between Guards and Triggers.
However, this guide will make this distinction to make it clear that signals should not be "sticking". Sticking signals are signals that are active for multiple candles. This can lead into entering a signal late (right before the signal disappears - which means that the chance of success is a lot lower than right at the beginning).

Hyper-optimization will, for each epoch round, pick one trigger and possibly multiple guards.

Exit signal optimization

Similar to the entry-signal above, exit-signals can also be optimized. Place the corresponding settings into the following methods

- Define the parameters at the class level hyperopt shall be optimizing, either naming them `sell_*`, or by explicitly defining `space='sell'`.
- Within `populate_exit_trend()` - use defined parameter values instead of raw constants.

The configuration and rules are the same than for buy signals.

Solving a Mystery

Let's say you are curious: should you use MACD crossings or lower Bollinger Bands to trigger your long entries. And you also wonder should you use RSI or ADX to help with those decisions. If you decide to use RSI or ADX, which values should I use for them?

So let's use hyperparameter optimization to solve this mystery.

Defining indicators to be used

We start by calculating the indicators our strategy is going to use.

```
class MyAwesomeStrategy(IStrategy):

    def populate_indicators(self, dataframe: DataFrame, metadata: dict) -> DataFrame:
        """
        Generate all indicators used by the strategy
        """
        dataframe['adx'] = ta.ADX(dataframe)
        dataframe['rsi'] = ta.RSI(dataframe)
        macd = ta.MACD(dataframe)
        dataframe['macd'] = macd['macd']
        dataframe['macdsignal'] = macd['macdsignal']
        dataframe['macdhist'] = macd['macdhist']
```

```
        bollinger = ta.BBANDS(dataframe, timeperiod=20, nbdevup=2.0, nbdevdn=2.0)
        dataframe['bb_lowerband'] = bollinger['lowerband']
        dataframe['bb_middleband'] = bollinger['middleband']
        dataframe['bb_upperband'] = bollinger['upperband']
        return dataframe
```

Hyperoptable parameters

We continue to define hyperoptable parameters:

```
class MyAwesomeStrategy(IStrategy):
    buy_adx = DecimalParameter(20, 40, decimals=1, default=30.1, space="buy")
    buy_rsi = IntParameter(20, 40, default=30, space="buy")
    buy_adx_enabled = BooleanParameter(default=True, space="buy")
    buy_rsi_enabled = CategoricalParameter([True, False], default=False, space="buy")
    buy_trigger = CategoricalParameter(["bb_lower", "macd_cross_signal"],
default="bb_lower", space="buy")
```

The above definition says: I have five parameters I want to randomly combine to find the best combination.
`buy_rsi` is an integer parameter, which will be tested between 20 and 4
`buy_adx` is a decimal parameter, which will be evaluated between 20 and 40 with 1 decimal place (so values are 20.1, 20.2, ...). This space has a size of 200.

Then we have three category variables. First two are either `True` or `False`. We use these to either enable or disable the ADX and RSI guards. The last one we call `trigger` and use it to decide which buy trigger we want to use.

Parameter space assignment

Parameters must either be assigned to a variable named `buy_*` or `sell_*` - or contain `space='buy'` | `space='sell'` to be assigned to a space correctly. If no parameter is available for a space, you'll receive the error that no space was found when running hyperopt.
Parameters with unclear space (e.g. `adx_period = IntParameter(4, 24, default=14)` - no explicit nor implicit space) will not be detected and will therefore be ignored.

So let's write the buy strategy using these values:

```
    def populate_entry_trend(self, dataframe: DataFrame, metadata: dict) -> DataFrame:
        conditions = []
        # GUARDS AND TRENDS
        if self.buy_adx_enabled.value:
            conditions.append(dataframe['adx'] > self.buy_ad
        if self.buy_rsi_enabled.value:
            conditions.append(dataframe['rsi'] < self.buy_rsi.value)
```

```
        # TRIGGERS
        if self.buy_trigger.value == 'bb_lower':
            conditions.append(dataframe['close'] < dataframe['bb_lowerband'])
        if self.buy_trigger.value == 'macd_cross_signal':
            conditions.append(qtpylib.crossed_above(
                dataframe['macd'], dataframe['macdsignal']
            ))

        # Check that volume is not 0
        conditions.append(dataframe['volume'] > 0)

        if conditions:
            dataframe.loc[
                reduce(lambda x, y: x & y, conditions),
                'enter_long'] = 1

        return dataframe
```

Hyperopt will now call `populate_entry_trend()` many times (`epochs`) with different value combinations. It will use the given historical data and simulate buys based on the buy signals generated with the above function.

Based on the results, hyperopt will tell you which parameter combinatio

on the configured loss function).

Note

The above setup expects to find ADX, RSI and Bollinger Bands in the populated indicators. When you want to test an indicator that isn't used by the bot currently, remember to add it to the `populate_indicators()` method in your strategy or hyperopt file.

Parameter types

There are four parameter types each suited for different purposes.

- `IntParameter` - defines an integral parameter with upper and lower boundaries of search space.
- `DecimalParameter` - defines a floating point parameter with a limited number of decimals (default 3). Should be preferred instead of `RealParameter` in most cases.
- `RealParameter` - defines a floating point parameter with upper and lower boundaries and no precision limit. Rarely used as it creates a space with a near infinite number of possibilities.
- `CategoricalParameter` - defines a parameter with a predetermined

- `BooleanParameter` - Shorthand for `CategoricalParameter([True, False])` - great for "enable" parameters.

Parameter options

There are two parameter options that can help you to quickly test various ideas:

- `optimize` - when set to `False`, the parameter will not be included in optimization process. (Default: True)
- `load` - when set to `False`, results of a previous hyperopt run (in `buy_params` and `sell_params` either in your strategy or the JSON output file) will not be used as the starting value for subsequent hyperopts. The default value specified in the parameter will be used instead. (Default: True)

Effects of `load=False` on backtesting

Be aware that setting the `load` option to `False` will mean backtesting will also use the default value specified in the parameter and *not* the value found through hyperoptimisation.

Warning

Hyperoptable parameters cannot be used in `populate_indicators` - as h
indicators for each epoch, so the starting value would be used in this case.

Optimizing an indicator parameter

Assuming you have a simple strategy in mind - a EMA cross strategy (2 Moving averages crossing) - and you'd like to find the ideal parameters for this strategy. By default, we assume a stoploss of 5% - and a take-profit (`minimal_roi`) of 10% - which means freqtrade will sell the trade once 10% profit has been reached.

```
from pandas import DataFrame
from functools import reduce

import talib.abstract as ta

from freqtrade.strategy import (BooleanParameter, CategoricalParameter, DecimalParameter,
                                IStrategy, IntParameter)
import freqtrade.vendor.qtpylib.indicators as qtpylib

class MyAwesomeStrategy(IStrategy):
    stoploss = -0.05
    timeframe = '15m'
    minimal_roi = {
        "0":  0.10
    }
    # Define the parameter spaces
    buy_ema_short = IntParameter(3, 50, default=5)
    buy_ema_long = IntParameter(15, 200, default=50)
```

```
    def populate_indicators(self, dataframe: DataFrame, metadata: dict) -> DataFrame:
        """Generate all indicators used by the strategy"""

        # Calculate all ema_short values
        for val in self.buy_ema_short.range:
            dataframe[f'ema_short_{val}'] = ta.EMA(dataframe, timeperiod=val)

        # Calculate all ema_long values
        for val in self.buy_ema_long.range:
            dataframe[f'ema_long_{val}'] = ta.EMA(dataframe, timeperiod=val)

        return dataframe

    def populate_entry_trend(self, dataframe: DataFrame, metadata: dict) -> DataFrame:
        conditions = []
        conditions.append(qtpylib.crossed_above(
                dataframe[f'ema_short_{self.buy_ema_short.value}'],
dataframe[f'ema_long_{self.buy_ema_long.value}']
            ))

        # Check that volume is not 0
        conditions.append(dataframe['volume'] > 0)

        if conditions:
            dataframe.loc[
                reduce(lambda x, y: x & y, conditions),
                'enter_long'] = 1
        return dataframe
```

```
    def populate_exit_trend(self, dataframe: DataFrame, metadata: dict) -> DataFrame:
        conditions = []
        conditions.append(qtpylib.crossed_above(
                dataframe[f'ema_long_{self.buy_ema_long.value}'],
dataframe[f'ema_short_{self.buy_ema_short.value}']
            ))

        # Check that volume is not 0
        conditions.append(dataframe['volume'] > 0)

        if conditions:
            dataframe.loc[
                reduce(lambda x, y: x & y, conditions),
                'exit_long'] = 1
        return dataframe
```

Breaking it down:

Using `self.buy_ema_short.range` will return a range object containing all entries between the Parameters low and high value. In this case (`IntParameter(3, 50, default=5)`), the loop would run for all numbers between 3 and 50 (`[3, 4, 5, ... 49, 50]`). By using this in a loop, hy ;
(`['buy_ema_3', 'buy_ema_4', ... , 'buy_ema_50']`).

Hyperopt itself will then use the selected value to create the buy and sell signals.

While this strategy is most likely too simple to provide consistent profit, it should serve as an example how optimize indicator parameters.

Note

`self.buy_ema_short.range` will act differently between hyperopt and other modes. For hyperopt, the above example may generate 48 new columns, however for all other modes (backtesting, dry/live), it will only generate the column for the selected value. You should therefore avoid using the resulting column with explicit values (values other than `self.buy_ema_short.value`).

Note

`range` property may also be used with `DecimalParameter` and `CategoricalParameter`. `RealParameter` does not provide this property due to infinite search space.

Performance tip

During normal hyperopting, indicators are calculated once and supplied to each epoch, linearly increasing RAM usage as a factor of increasing cores. As this also has performance implications, there are two alternatives to reduce RAM usage

- Move `ema_short` and `ema_long` calculations from `populate_indicators()` to `populate_entry_trend()`. Since `populate_entry_trend()` will be calculated every epoch, you don't need to use `.range` functionality.
- hyperopt provides `--analyze-per-epoch` which will move the execution of `populate_indicators()` to the epoch process, calculating a single value per parameter per epoch instead of using the `.range` functionality. In this case, `.range` functionality will only return the actually used value.

These alternatives will reduce RAM usage, but increase CPU usage. However, your hyperopting run will be less likely to fail due to Out Of Memory (OOM) issues.

Whether you are using `.range` functionality or the alternatives above, you should try to use space ranges as small as possible since this will improve CPU/RAM usage.

Optimizing protections

Freqtrade can also optimize protections. How you optimize protections is up to you, and the following should be considered as example only.

The strategy will simply need to define the "protections" entry as property returning a list of protection configurations.

```
from pandas import DataFrame
from functools import reduce

import talib.abstract as ta

from freqtrade.strategy import (BooleanParameter, CategoricalParameter, DecimalParameter,
                                IStrategy, IntParameter)
import freqtrade.vendor.qtpylib.indicators as qtpylib

class MyAwesomeStrategy(IStrategy):
    stoploss = -0.05
    timeframe = '15m'
    # Define the parameter spaces
    cooldown_lookback = IntParameter(2, 48, default=5, space="protection", optimize=True)
    stop_duration = IntParameter(12, 200, default=5, space="protection", optimize=True)
    use_stop_protection = BooleanParameter(default=True, spa
optimize=True)


    @property
```

```
    def protections(self):
        prot = []

        prot.append({
            "method": "CooldownPeriod",
            "stop_duration_candles": self.cooldown_lookback.value
        })
        if self.use_stop_protection.value:
            prot.append({
                "method": "StoplossGuard",
                "lookback_period_candles": 24 * 3,
                "trade_limit": 4,
                "stop_duration_candles": self.stop_duration.value,
                "only_per_pair": False
            })

        return prot

    def populate_indicators(self, dataframe: DataFrame, metadata: dict) -> DataFrame:
        # ...
```

You can then run hyperopt as follows: `freqtrade hyperopt --hyperopt-loss SharpeHyperOptLossDaily --strategy MyAwesomeStrategy --spaces protection`

Note

The protection space is not part of the default space, and is only available with the Parameters Hyperopt interface, not with the legacy hyperopt interface (which required separate hyperopt files). Freqtrade will also automatically change the "--enable-protections" flag if the protection space is selected.

Warning

If protections are defined as property, entries from the configuration will be ignored. It is therefore recommended to not define protections in the configuration.

Migrating from previous property setups

A migration from a previous setup is pretty simple, and can be accomplished by converting the protections entry to a property. In simple terms, the following configuration will be converted to the below.


```
class MyAwesomeStrategy(IStrategy):
    protections = [
        {
            "method": "CooldownPeriod",
```

```
            "stop_duration_candles": 4
        }
    ]
```

Result

```
class MyAwesomeStrategy(IStrategy):

    @property
    def protections(self):
        return [
            {
                "method": "CooldownPeriod",
                "stop_duration_candles": 4
            }
        ]
```

You will then obviously also change potential interesting entries to parameters to allow hyper-optimization.

Optimizing `max_entry_position_adjustment`

While `max_entry_position_adjustment` is not a separate space, it can property approach shown above.

```
from pandas import DataFrame
from functools import reduce

import talib.abstract as ta

from freqtrade.strategy import (BooleanParameter, CategoricalParameter, DecimalParameter,
                                IStrategy, IntParameter)
import freqtrade.vendor.qtpylib.indicators as qtpylib

class MyAwesomeStrategy(IStrategy):
    stoploss = -0.05
    timeframe = '15m'

    # Define the parameter spaces
    max_epa = CategoricalParameter([-1, 0, 1, 3, 5, 10], default=1, space="buy",
optimize=True)

    @property
    def max_entry_position_adjustment(self):
        return self.max_epa.value


    def populate_indicators(self, dataframe: DataFrame, meta
        # ...
```

Using `IntParameter`

You can also use the `IntParameter` for this optimization, but you must explicitly return an integer:

```
max_epa = IntParameter(-1, 10, default=1, space="buy", optimize=True)

@property
def max_entry_position_adjustment(self):
    return int(self.max_epa.value)
```

Loss-functions

Each hyperparameter tuning requires a target. This is usually defined as a loss function (sometimes also called objective function), which should decrease for more desirable results, and increase for bad results.

A loss function must be specified via the `--hyperopt-loss <Class-name>` argument (or optionally via the configuration under the `"hyperopt_loss"` key). This class should be in its own file within the `user_data/hyperopts/` directory.

Currently, the following loss functions are builtin:

- `ShortTradeDurHyperOptLoss` - (default legacy Freqtrade hyperoptimization loss function) - Mostly for short trade duration and avoiding losses.
- `OnlyProfitHyperOptLoss` - takes only amount of profit into consideration.
- `SharpeHyperOptLoss` - optimizes Sharpe Ratio calculated on trade returns relative to standard deviation.
- `SharpeHyperOptLossDaily` - optimizes Sharpe Ratio calculated on **daily** trade returns relative to standard deviation.
- `SortinoHyperOptLoss` - optimizes Sortino Ratio calculated on trade returns relative to **downside** standard deviation.
- `SortinoHyperOptLossDaily` - optimizes Sortino Ratio calculated on **daily** trade returns relative to **downside** standard deviation.
- `MaxDrawDownHyperOptLoss` - Optimizes Maximum absolute drawdown.
- `MaxDrawDownRelativeHyperOptLoss` - Optimizes both maximum absolute drawdown while also adjusting for maximum relative drawdown.
- `CalmarHyperOptLoss` - Optimizes Calmar Ratio calculated on trade
- `ProfitDrawDownHyperOptLoss` - Optimizes by max Profit & min Drawdown objective. `DRAWDOWN_MULT` variable within the hyperoptloss file can be adjusted to be stricter or more flexible on drawdown

purposes.

Creation of a custom loss function is covered in the Advanced Hyperopt part of the documentation.

Execute Hyperopt

Once you have updated your hyperopt configuration you can run it. Because hyperopt tries a lot of combinations to find the best parameters it will take time to get a good result.

We strongly recommend to use `screen` or `tmux` to prevent any connection loss.

```
freqtrade hyperopt --config config.json --hyperopt-loss <hyperoptlossname> --strategy
<strategyname> -e 500 --spaces all
```

The `-e` option will set how many evaluations hyperopt will do. Since hyperopt uses Bayesian search, running too many epochs at once may not produce greater results. Experience has shown that best results are usually not improving much after 500-1000 epochs.
Doing multiple runs (executions) with a few 1000 epochs and different r[illegible] will most likely produce different results.

The `--spaces all` option determines that all possible parameters should be optimized. Possibilities are

listed below.

Note

Hyperopt will store hyperopt results with the timestamp of the hyperopt start time. Reading commands (`hyperopt-list`, `hyperopt-show`) can use `--hyperopt-filename <filename>` to read and display older hyperopt results. You can find a list of filenames with `ls -l user_data/hyperopt_results/`.

Execute Hyperopt with different historical data source

If you would like to hyperopt parameters using an alternate historical data set that you have on-disk, use the `--datadir PATH` option. By default, hyperopt uses data from directory `user_data/data`.

Running Hyperopt with a smaller test-set

Use the `--timerange` argument to change how much of the test-set you want to use. For example, to use one month of data, pass `--timerange 20210101-20210201` (from janua

hyperopt call.

Full command:

```
freqtrade hyperopt --strategy <strategyname> --timerange 20210101-20210201
```

Running Hyperopt with Smaller Search Space

Use the `--spaces` option to limit the search space used by hyperopt. Letting Hyperopt optimize everything is a huuuuge search space. Often it might make more sense to start by just searching for initial buy algorithm. Or maybe you just want to optimize your stoploss or roi table for that awesome new buy strategy you have.

Legal values are:

- `all`: optimize everything
- `buy`: just search for a new buy strategy
- `sell`: just search for a new sell strategy
- `roi`: just optimize the minimal profit table for your strategy
- `stoploss`: search for the best stoploss value
- `trailing`: search for the best trailing stop values
- `trades`: search for the best max open trades values

- `protection`: search for the best protection parameters (read the protections section on how to properly define these)
- `default`: `all` except `trailing` and `protection`
- space-separated list of any of the above values for example `--spaces roi stoploss`

The default Hyperopt Search Space, used when no `--space` command line option is specified, does not include the `trailing` hyperspace. We recommend you to run optimization for the `trailing` hyperspace separately, when the best parameters for other hyperspaces were found, validated and pasted into your custom strategy.

Understand the Hyperopt Result

Once Hyperopt is completed you can use the result to update your strategy. Given the following result from hyperopt:

```
Best result:

    44/100:    135 trades. Avg profit  0.57%. Total profit
duration 180.4 mins. Objective: 1.94367

    # Buy hyperspace params:
```

```
    buy_params = {
        'buy_adx': 44,
        'buy_rsi': 29,
        'buy_adx_enabled': False,
        'buy_rsi_enabled': True,
        'buy_trigger': 'bb_lower'
    }
```

You should understand this result like:

- The buy trigger that worked best was `bb_lower`.
- You should not use ADX because `'buy_adx_enabled': False`.
- You should **consider** using the RSI indicator (`'buy_rsi_enabled': True`) and the best value is `29.0` (`'buy_rsi': 29.0`)

Automatic parameter application to the strategy

When using Hyperoptable parameters, the result of your hyperopt-run will be written to a json file next to your strategy (so for `MyAwesomeStrategy.py`, the file would be `MyAwesc`
This file is also updated when using the `hyperopt-show` sub-command
provided to either of the 2 commands.

Your strategy class can also contain these results explicitly. Simply copy hyperopt results block and paste them at class level, replacing old parameters (if any). New parameters will automatically be loaded next time strategy is executed.

Transferring your whole hyperopt result to your strategy would then look like:

```
class MyAwesomeStrategy(IStrategy):
    # Buy hyperspace params:
    buy_params = {
        'buy_adx': 44,
        'buy_rsi': 29,
        'buy_adx_enabled': False,
        'buy_rsi_enabled': True,
        'buy_trigger': 'bb_lower'
    }
```

Note

Values in the configuration file will overwrite Parameter-file level parameters - and both will overwrite parameters within the strategy. The prevalence is therefore: config > parar parameter default

Understand Hyperopt ROI results

If you are optimizing ROI (i.e. if optimization search-space contains 'all', 'default' or 'roi'), your result will look as follows and include a ROI table:

```
Best result:

    44/100:    135 trades. Avg profit  0.57%. Total profit  0.03871918 BTC (0.7722%). Avg
duration 180.4 mins. Objective: 1.94367

    # ROI table:
    minimal_roi = {
        0: 0.10674,
        21: 0.09158,
        78: 0.03634,
        118: 0
    }
```

In order to use this best ROI table found by Hyperopt in backtesting and for live trades/dry-run, copy-paste it as the value of the `minimal_roi` attribute of your custom strategy:


```
    # Minimal ROI designed for the strategy.
    # This attribute will be overridden if the config file contains "minimal_roi"
    minimal_roi = {
        0: 0.10674,
```

```
        21: 0.09158,
        78: 0.03634,
        118: 0
    }
```

As stated in the comment, you can also use it as the value of the `minimal_roi` setting in the configuration file.

Default ROI Search Space

If you are optimizing ROI, Freqtrade creates the 'roi' optimization hyperspace for you -- it's the hyperspace of components for the ROI tables. By default, each ROI table generated by the Freqtrade consists of 4 rows (steps). Hyperopt implements adaptive ranges for ROI tables with ranges for values in the ROI steps that depend on the timeframe used. By default the values vary in the following ranges (for some of the most used timeframes, values are rounded to 3 digits after the decimal point):

# step	1m		5m		1h	
1	0	0.011...0.119	0	0		8...(
2	2...8	0.007...0.042	10...40	0.02...0.11	120...480	0.045...(

# step	1m		5m		1h	
3	4...20	0.003...0.015	20...100	0.01...0.04	240...1200	0.022...(
4	6...44	0.0	30...220	0.0	360...2640	0.0

These ranges should be sufficient in most cases. The minutes in the steps (ROI dict keys) are scaled linearly depending on the timeframe used. The ROI values in the steps (ROI dict values) are scaled logarithmically depending on the timeframe used.

If you have the `generate_roi_table()` and `roi_space()` methods in your custom hyperopt, remove them in order to utilize these adaptive ROI tables and the ROI hyperoptimization space generated by Freqtrade by default.

Override the `roi_space()` method if you need components of the ROI tables to vary in other ranges. Override the `generate_roi_table()` and `roi_space()` methods and implement your own custom approach for generation of the ROI tables during hyperoptimization if you need a d[illegible] or other amount of rows (steps).

A sample for these methods can be found in the overriding pre-defined spaces section.

Reduced search space

To limit the search space further, Decimals are limited to 3 decimal places (a precision of 0.001). This is usually sufficient, every value more precise than this will usually result in overfitted results. You can however overriding pre-defined spaces to change this to your needs.

Understand Hyperopt Stoploss results

If you are optimizing stoploss values (i.e. if optimization search-space contains 'all', 'default' or 'stoploss'), your result will look as follows and include stoploss:

```
Best result:

    44/100:    135 trades. Avg profit  0.57%. Total profit  0.03871918 BTC (0.7722%). Avg
duration 180.4 mins. Objective: 1.94367

    # Buy hyperspace params:
    buy_params = {
        'buy_adx': 44,
        'buy_rsi': 29,
        'buy_adx_enabled': False,
        'buy_rsi_enabled': True,
        'buy_trigger': 'bb_lower'
```

```
    }

    stoploss: -0.27996
```

In order to use this best stoploss value found by Hyperopt in backtesting and for live trades/dry-run, copy-paste it as the value of the `stoploss` attribute of your custom strategy:

```
    # Optimal stoploss designed for the strategy
    # This attribute will be overridden if the config file contains "stoploss"
    stoploss = -0.27996
```

As stated in the comment, you can also use it as the value of the `stoploss` setting in the configuration file.

Default Stoploss Search Space

If you are optimizing stoploss values, Freqtrade creates the 'stoploss' optimization hyperspace for you. By default, the stoploss values in that hyperspace vary in the range -0.35...-0.02, which is sufficient in most cases.

If you have the `stoploss_space()` method in your custom hyperopt file [illegible] hyperoptimization space generated by Freqtrade by default.

Override the `stoploss_space()` method and define the desired range in it if you need stoploss values to

vary in other range during hyperoptimization. A sample for this method can be found in the overriding pre-defined spaces section.

Reduced search space

To limit the search space further, Decimals are limited to 3 decimal places (a precision of 0.001). This is usually sufficient, every value more precise than this will usually result in overfitted results. You can however overriding pre-defined spaces to change this to your needs.

Understand Hyperopt Trailing Stop results

If you are optimizing trailing stop values (i.e. if optimization search-space contains 'all' or 'trailing'), your result will look as follows and include trailing stop parameters:

```
Best result:

    45/100:    606 trades. Avg profit  1.04%. Total profit  0.31555614 BTC ( 630.48%). Avg
duration 150.3 mins. Objective: -1.10161

    # Trailing stop:
    trailing_stop = True
    trailing_stop_positive = 0.02001
```

```
    trailing_stop_positive_offset = 0.06038
    trailing_only_offset_is_reached = True
```

In order to use these best trailing stop parameters found by Hyperopt in backtesting and for live trades/dry-run, copy-paste them as the values of the corresponding attributes of your custom strategy:

```
    # Trailing stop
    # These attributes will be overridden if the config file contains corresponding
values.
    trailing_stop = True
    trailing_stop_positive = 0.02001
    trailing_stop_positive_offset = 0.06038
    trailing_only_offset_is_reached = True
```

As stated in the comment, you can also use it as the values of the corresponding settings in the configuration file.

Default Trailing Stop Search Space

If you are optimizing trailing stop values, Freqtrade creates the 'trailing' optimization hyperspace for you. By default, the `trailing_stop` parameter is always set to True in that hype

`trailing_only_offset_is_reached` vary between True and False, the v

`trailing_stop_positive` and `trailing_stop_positive_offset` parameters vary in the ranges 0.02...0.35

and 0.01...0.1 correspondingly, which is sufficient in most cases.

Override the `trailing_space()` method and define the desired range in it if you need values of the trailing stop parameters to vary in other ranges during hyperoptimization. A sample for this method can be found in the overriding pre-defined spaces section.

Reduced search space

To limit the search space further, Decimals are limited to 3 decimal places (a precision of 0.001). This is usually sufficient, every value more precise than this will usually result in overfitted results. You can however overriding pre-defined spaces to change this to your needs.

Reproducible results

The search for optimal parameters starts with a few (currently 30) random combinations in the hyperspace of parameters, random Hyperopt epochs. These random epochs are marked with an asterisk character (`*`) in the first column in the Hyperopt output.

The initial state for generation of these random values (random state) is controlled by the value of the `--random-state` command line option. You can set it to some arbitrary value of your choice to obtain

reproducible results.

If you have not set this value explicitly in the command line options, Hyperopt seeds the random state with some random value for you. The random state value for each Hyperopt run is shown in the log, so you can copy and paste it into the `--random-state` command line option to repeat the set of the initial random epochs used.

If you have not changed anything in the command line options, configuration, timerange, Strategy and Hyperopt classes, historical data and the Loss Function -- you should obtain same hyper-optimization results with same random state value used.

Output formatting

By default, hyperopt prints colorized results -- epochs with positive profit are printed in the green color. This highlighting helps you find epochs that can be interesting for later analysis. Epochs with zero total profit or with negative profits (losses) are printed in the normal color. If you do not need colorization of results (for instance, when you are redirecting hyperopt output to a file) you can switch colorization off by specifying the `--no-color` option in the command line.

You can use the `--print-all` command line option if you would like to see all results in the hyperopt output, not only the best ones. When `--print-all` is used, current best results are also colorized by default

-- they are printed in bold (bright) style. This can also be switched off with the `--no-color` command line option.

Windows and color output

Windows does not support color-output natively, therefore it is automatically disabled. To have color-output for hyperopt running under windows, please consider using WSL.

Position stacking and disabling max market positions

In some situations, you may need to run Hyperopt (and Backtesting) with the `--eps` / `--enable-position-staking` and `--dmmp` / `--disable-max-market-positions` arguments.

By default, hyperopt emulates the behavior of the Freqtrade Live Run/Dry Run, where only one open trade is allowed for every traded pair. The total number of trades open for all pairs is also limited by the `max_open_trades` setting. During Hyperopt/Backtesting this may lead to some potential trades to be hidden (or masked) by previously open trades.

The `--eps` / `--enable-position-stacking` argument allows emulation of buying the same pair multiple times, while `--dmmp` / `--disable-max-market-positions` disables applying `max_open_trades` during

Hyperopt/Backtesting (which is equal to setting `max_open_trades` to a very high number).

> **Note**
>
> Dry/live runs will **NOT** use position stacking - therefore it does make sense to also validate the strategy without this as it's closer to reality.

You can also enable position stacking in the configuration file by explicitly setting `"position_stacking"=true`.

Out of Memory errors

As hyperopt consumes a lot of memory (the complete data needs to be in memory once per parallel backtesting process), it's likely that you run into "out of memory" errors. To combat these, you have multiple options:

- Reduce the amount of pairs.
- Reduce the timerange used (`--timerange <timerange>`).
- Avoid using `--timeframe-detail` (this loads a lot of additional data into memory).

- Reduce the number of parallel processes (`-j <n>`).
- Increase the memory of your machine.
- Use `--analyze-per-epoch` if you're using a lot of parameters with `.range` functionality.

The objective has been evaluated at this point before.

If you see `The objective has been evaluated at this point before.` - then this is a sign that your space has been exhausted, or is close to that. Basically all points in your space have been hit (or a local minima has been hit) - and hyperopt does no longer find points in the multi-dimensional space it did not try yet. Freqtrade tries to counter the "local minima" problem by using new, randomized points in this case.

Example:

```
buy_ema_short = IntParameter(5, 20, default=10, space="buy", optimize=True)
# This is the only parameter in the buy space
```

The `buy_ema_short` space has 15 possible values (`5, 6, ... 19, 20`) if [illegible] space, hyperopt will only have 15 values to try before running out of opti[illegible] be aligned to the possible values - or you should be ready to interrupt a run if you norice a lot of `The objective has been evaluated at this point before.` warnings.

Show details of Hyperopt results

After you run Hyperopt for the desired amount of epochs, you can later list all results for analysis, select only best or profitable once, and show the details for any of the epochs previously evaluated. This can be done with the `hyperopt-list` and `hyperopt-show` sub-commands. The usage of these sub-commands is described in the Utils chapter.

Validate backtesting results

Once the optimized strategy has been implemented into your strategy, you should backtest this strategy to make sure everything is working as expected.

To achieve same the results (number of trades, their durations, profit, etc.) as during Hyperopt, please use the same configuration and parameters (timerange, timeframe, ...) used for hyperopt `--dmmp`/`--disable-max-market-positions` and `--eps`/`--enable-position-stacking` for Backtesting.

Why do my backtest results not match my hyperopt results?

Should results not match, check the following factors:

- You may have added parameters to hyperopt in `populate_indicators()` where they will be calculated

only once **for all epochs**. If you are, for example, trying to optimise multiple SMA timeperiod values, the hyperoptable timeperiod parameter should be placed in `populate_entry_trend()` which is calculated every epoch. See Optimizing an indicator parameter.

- If you have disabled the auto-export of hyperopt parameters into the JSON parameters file, double-check to make sure you transferred all hyperopted values into your strategy correctly.
- Check the logs to verify what parameters are being set and what values are being used.
- Pay special care to the stoploss, max_open_trades and trailing stoploss parameters, as these are often set in configuration files, which override changes to the strategy. Check the logs of your backtest to ensure that there were no parameters inadvertently set by the configuration (like `stoploss`, `max_open_trades` or `trailing_stop`).
- Verify that you do not have an unexpected parameters JSON file overriding the parameters or the default hyperopt settings in your strategy.
- Verify that any protections that are enabled in backtesting are also enabled when hyperopting, and vice versa. When using `--space protection`, protections are auto-enabled for hyperopting.