strategy_analysis_example

September 29, 2021

1 Strategy analysis example

Debugging a strategy can be time-consuming. Freqtrade offers helper functions to visualize raw data. The following assumes you work with SampleStrategy, data for 5m timeframe from Binance and have downloaded them into the data directory in the default location.

1.1 Setup

[ ]:
```
from pathlib import Path
from freqtrade.configuration import Configuration

# Customize these according to your needs.

# Initialize empty configuration object
config = Configuration.from_files([])
# Optionally, use existing configuration file
# config = Configuration.from_files(["config.json"])

# Define some constants
config["timeframe"] = "5m"
# Name of the strategy class
config["strategy"] = "SampleStrategy"
# Location of the data
data_location = Path(config['user_data_dir'], 'data', 'binance')
# Pair to analyze - Only use one pair here
pair = "BTC/USDT"
```

[ ]:
```
# Load data using values set above
from freqtrade.data.history import load_pair_history

candles = load_pair_history(datadir=data_location,
                            timeframe=config["timeframe"],
                            pair=pair,
                            data_format = "hdf5",
                            )

# Confirm success
```

```
print("Loaded " + str(len(candles)) + f" rows of data for {pair} from␣
 ↪{data_location}")
candles.head()
```

1.2 Load and run strategy

- Rerun each time the strategy file is changed

[ ]:
```
# Load strategy using values set above
from freqtrade.resolvers import StrategyResolver
strategy = StrategyResolver.load_strategy(config)

# Generate buy/sell signals using strategy
df = strategy.analyze_ticker(candles, {'pair': pair})
df.tail()
```

1.2.1 Display the trade details

- Note that using `data.head()` would also work, however most indicators have some "startup" data at the top of the dataframe.
- Some possible problems
  - Columns with NaN values at the end of the dataframe
  - Columns used in `crossed*()` functions with completely different units
- Comparison with full backtest
  - having 200 buy signals as output for one pair from `analyze_ticker()` does not necessarily mean that 200 trades will be made during backtesting.
  - Assuming you use only one condition such as, `df['rsi'] < 30` as buy condition, this will generate multiple "buy" signals for each pair in sequence (until rsi returns > 29). The bot will only buy on the first of these signals (and also only if a trade-slot ("max_open_trades") is still available), or on one of the middle signals, as soon as a "slot" becomes available.

[ ]:
```
# Report results
print(f"Generated {df['buy'].sum()} buy signals")
data = df.set_index('date', drop=False)
data.tail()
```

1.3 Load existing objects into a Jupyter notebook

The following cells assume that you have already generated data using the cli.
They will allow you to drill deeper into your results, and perform analysis which otherwise would make the output very difficult to digest due to information overload.

1.3.1 Load backtest results to pandas dataframe

Analyze a trades dataframe (also used below for plotting)

[ ]:
```
from freqtrade.data.btanalysis import load_backtest_data, load_backtest_stats

# if backtest_dir points to a directory, it'll automatically load the last␣
 ↪backtest file.
backtest_dir = config["user_data_dir"] / "backtest_results"
# backtest_dir can also point to a specific file
# backtest_dir = config["user_data_dir"] / "backtest_results/
 ↪backtest-result-2020-07-01_20-04-22.json"
```

[ ]:
```
# You can get the full backtest statistics by using the following command.
# This contains all information used to generate the backtest result.
stats = load_backtest_stats(backtest_dir)

strategy = 'SampleStrategy'
# All statistics are available per strategy, so if `--strategy-list` was used␣
 ↪during backtest, this will be reflected here as well.
# Example usages:
print(stats['strategy'][strategy]['results_per_pair'])
# Get pairlist used for this backtest
print(stats['strategy'][strategy]['pairlist'])
# Get market change (average change of all pairs from start to end of the␣
 ↪backtest period)
print(stats['strategy'][strategy]['market_change'])
# Maximum drawdown ()
print(stats['strategy'][strategy]['max_drawdown'])
# Maximum drawdown start and end
print(stats['strategy'][strategy]['drawdown_start'])
print(stats['strategy'][strategy]['drawdown_end'])


# Get strategy comparison (only relevant if multiple strategies were compared)
print(stats['strategy_comparison'])
```

[ ]:
```
# Load backtested trades as dataframe
trades = load_backtest_data(backtest_dir)

# Show value-counts per pair
trades.groupby("pair")["sell_reason"].value_counts()
```

1.4 Plotting daily profit / equity line

[ ]:
```
# Plotting equity line (starting with 0 on day 1 and adding daily profit for␣
 ↪each backtested day)

from freqtrade.configuration import Configuration
from freqtrade.data.btanalysis import load_backtest_data, load_backtest_stats
```

```
import plotly.express as px
import pandas as pd

# strategy = 'SampleStrategy'
# config = Configuration.from_files(["user_data/config.json"])
# backtest_dir = config["user_data_dir"] / "backtest_results"

stats = load_backtest_stats(backtest_dir)
strategy_stats = stats['strategy'][strategy]

dates = []
profits = []
for date_profit in strategy_stats['daily_profit']:
    dates.append(date_profit[0])
    profits.append(date_profit[1])

equity = 0
equity_daily = []
for daily_profit in profits:
    equity_daily.append(equity)
    equity += float(daily_profit)


df = pd.DataFrame({'dates': dates,'equity_daily': equity_daily})

fig = px.line(df, x="dates", y="equity_daily")
fig.show()
```

1.4.1 Load live trading results into a pandas dataframe

In case you did already some trading and want to analyze your performance

[ ]:
```
from freqtrade.data.btanalysis import load_trades_from_db

# Fetch trades from database
trades = load_trades_from_db("sqlite:///tradesv3.sqlite")

# Display results
trades.groupby("pair")["sell_reason"].value_counts()
```

1.5 Analyze the loaded trades for trade parallelism

This can be useful to find the best `max_open_trades` parameter, when used with backtesting in conjunction with `--disable-max-market-positions`.

`analyze_trade_parallelism()` returns a timeseries dataframe with an "open_trades" column, specifying the number of open trades for each candle.

[ ]:
```
from freqtrade.data.btanalysis import analyze_trade_parallelism

# Analyze the above
parallel_trades = analyze_trade_parallelism(trades, '5m')

parallel_trades.plot()
```

1.6 Plot results

Freqtrade offers interactive plotting capabilities based on plotly.

[ ]:
```
from freqtrade.plot.plotting import  generate_candlestick_graph
# Limit graph period to keep plotly quick and reactive

# Filter trades to one pair
trades_red = trades.loc[trades['pair'] == pair]

data_red = data['2019-06-01':'2019-06-10']
# Generate candlestick graph
graph = generate_candlestick_graph(pair=pair,
                                   data=data_red,
                                   trades=trades_red,
                                   indicators1=['sma20', 'ema50', 'ema55'],
                                   indicators2=['rsi', 'macd', 'macdsignal',␣
 ↪'macdhist']
                                  )
```

[ ]:
```
# Show graph inline
# graph.show()

# Render graph in a seperate window
graph.show(renderer="browser")
```

1.7 Plot average profit per trade as distribution graph

[ ]:
```
import plotly.figure_factory as ff

hist_data = [trades.profit_ratio]
group_labels = ['profit_ratio']  # name of the dataset

fig = ff.create_distplot(hist_data, group_labels,bin_size=0.01)
fig.show()
```

Feel free to submit an issue or Pull Request enhancing this document if you would like to share ideas on how to best analyze the data.