Simple Trading Strategies

Jeremy Melvin

Initial Data Survey

- Let's start with a brief summary of the datasets
- Variables: Date / Epoch / Symbol / Open / High / Low / Close / Volume
- Source: Gemini

Table: Brief Summary of Provided Datasets

Market	Candle Size	Start Candle	End Candle
BTC-USD	1 hr	10/08/15 13:00	09/30/21 00:00
ETH-USD	1 hr	05/09/16 13:00	09/30/21 00:00
LTC-USD	1 hr	10/16/18 13:00	09/30/21 00:00

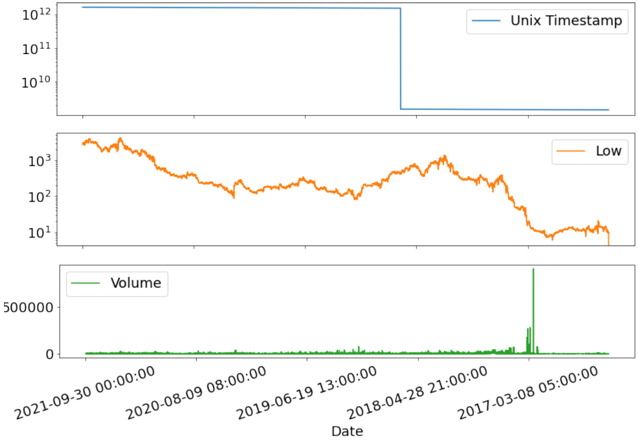
Initial Data Survey

- Build some initial infrastructure to manage the datasets
- Setup a function to clean the data

```python
# Manage the time series price data
class PriceTimeSeries:
    def __init__(self, asset, baseCurrency, timescale, csvFile=None, df=None):
        self.asset = asset
        self.baseCurrency = baseCurrency
        self.timescale = timescale
        if (csvFile):
            self.df = pd.read_csv(csvFile, skiprows=[0])
        else:
            self.df = df
        self.cleanData()

    # Any cleaning/initial processing of the data
    def cleanData(self):
        # Force dates to be datetime objects
        self.df['Date'] = pd.to_datetime(self.df['Date'])

        # Sort data ascending
        self.df = self.df.sort_values(by=['Date'], ignore_index=True)

        # Overwrite Unix Timestamp from Dates to handle s vs ms issue
        self.df['Unix Timestamp'] = self.df.Date.values.astype(np.int64) // 10 ** 9

```

Data Cleaning

- From visual inspection of data:
  - Unix Timestamps (s and ms)
  - Open/Low Missing Initial Data
  - Potential anomalies in Volumes
    - Verified in alternate candle data from source (CryptoDataDownload.com)
    - Price was steady / other sources (CMC) don't show volume spike
    - Will keep for now and use caution if volume data is used in analysis



Calculated Quantities

- Build out the dataset with some Technical Analysis quantites
  - RSI
  - Bollinger Bands
  - SMA/EMA

```python
# Technical Analysis indicators - by Darío López Padial
# https://pythonrepo.com/repo/bukosabino-ta-python-finance
from ta import add_all_ta_features
from ta.utils import dropna
from ta.volatility import BollingerBands
from ta.trend import SMAIndicator, EMAIndicator
from ta.momentum import RSIIndicator
```

```python
    # Add Relative Strength Index
    # default = 14 time units
    def addRSI(self, window=14):
        columnName = "RSI-" + str(window)
        self.df[columnName] = RSIIndicator(close=self.df['Close'], window=window, fillna=False).rsi()

    # Add Bollinger Bands Hi and Low Values
    # default = 20 time units, 2 standard deviations
    def addBB(self, window=20, stDev=2):
        bbData = BollingerBands(close=self.df['Close'], window=window, window_dev=stDev, fillna=False)

        columnName = "BB-" + str(window) + "-" + str(stDev) + "-hi"
        self.df[columnName] = bbData.bollinger_hband()
        columnName = "BB-" + str(window) + "-" + str(stDev) + "-low"
        self.df[columnName] = bbData.bollinger_lband()
        columnName = "BB-" + str(window) + "-" + str(stDev) + "-width"
        self.df[columnName] = bbData.bollinger_hband() - bbData.bollinger_lband()
```

Other Quantities

- Build out the dataset with some Technical Analysis quantites
  - RSI
  - Bollinger Bands
  - SMA/EMA

```python
# Technical Analysis indicators - by Darío López Padial
# https://pythonrepo.com/repo/bukosabino-ta-python-finance
from ta import add_all_ta_features
from ta.utils import dropna
from ta.volatility import BollingerBands
from ta.trend import SMAIndicator, EMAIndicator
from ta.momentum import RSIIndicator
```

```python
    # Add Relative Strength Index
    # default = 14 time units
    def addRSI(self, window=14):
        columnName = "RSI-" + str(window)
        self.df[columnName] = RSIIndicator(close=self.df['Close'], window=window, fillna=False).rsi()

    # Add Bollinger Bands Hi and Low Values
    # default = 20 time units, 2 standard deviations
    def addBB(self, window=20, stDev=2):
        bbData = BollingerBands(close=self.df['Close'], window=window, window_dev=stDev, fillna=False)

        columnName = "BB-" + str(window) + "-" + str(stDev) + "-hi"
        self.df[columnName] = bbData.bollinger_hband()
        columnName = "BB-" + str(window) + "-" + str(stDev) + "-low"
        self.df[columnName] = bbData.bollinger_lband()
        columnName = "BB-" + str(window) + "-" + str(stDev) + "-width"
        self.df[columnName] = bbData.bollinger_hband() - bbData.bollinger_lband()
```

Calculated Dependent Variables

- Need Response variables based on future data to train/test models

```
    # Add different ways to assess if this should of been a buy or sell
    def addEvals(self):
        # Option 1: If next candle is up, should of bought, if next candle down, should of sold
        # >0 --> "Buy", <0 --> "Sell" *** Keep raw value here, will map sign to Buy/Sell later
        columnName = "NextReturn"
        self.df[columnName] = (self.df['Close'].shift(-1) - self.df['Close'])/self.df['Close']

        # Option 2: Does close price go up 5% or down 5% first
        columnName = "FivePercent"
        self.df[columnName] = self.df.apply(fivePercent, axis=1, args=[self.df['Close']])
```

```
### HELPER FUNCTION
def fivePercent(row, column):
    incPrice = row['Close']*1.05
    decPrice = row['Close']*0.95
    indices = column[column >= incPrice].index
    priceUpIndex = indices[indices > row.name].min()
    indices = column[column <= decPrice].index
    priceDownIndex = indices[indices > row.name].min()

    if (priceDownIndex != priceDownIndex):
        if (priceUpIndex != priceUpIndex):
            return 'none'
        else:
            return 'buy'

    if (priceUpIndex != priceUpIndex):
```

Attempt to Build a Prediction Model

- Start with a regression model to predict Returns for next candle
- Independent variables have clear correlations, so concerns about multicollinearity
- Try to build-up a model using the BTC 4H dataset

	Variable	P-Value	R-Squared
3	logReturn	0.000010	0.001901
1	Volume	0.000012	0.001860
2	Return	0.000023	0.001738
4	RSI-14	0.001554	0.000971
8	RRt	0.006415	0.000721
48	d2EMA-100dt2	0.007158	0.000702
40	d2EMA-50dt2	0.007248	0.000699

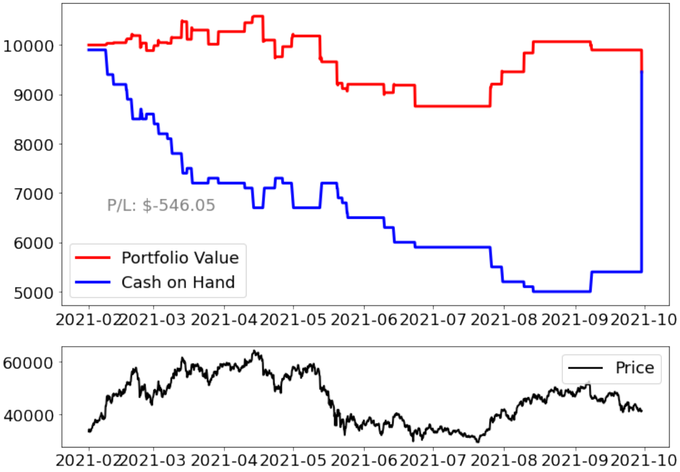
OLS Regression Results

Dep. Variable:	Signal	**R-squared:**	0.010
Model:	OLS	**Adj. R-squared:**	0.010
Method:	Least Squares	**F-statistic:**	20.94
Date:	Tue, 19 Oct 2021	**Prob (F-statistic):**	6.95e-21
Time:	20:10:46	**Log-Likelihood:**	27779.
No. Observations:	10308	**AIC:**	-5.555e+04
Df Residuals:	10302	**BIC:**	-5.550e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0079	0.001	-6.777	0.000	-0.010	-0.006
Volume	7.548e-07	1.81e-07	4.178	0.000	4.01e-07	1.11e-06
RSI-14	0.0002	2.25e-05	6.890	0.000	0.000	0.000
EMA-20-deviation	-0.0611	0.010	-5.896	0.000	-0.081	-0.041
dSMA-5dt	1.388e-05	2.58e-06	5.380	0.000	8.82e-06	1.89e-05
logReturn	-0.0520	0.011	-4.711	0.000	-0.074	-0.030

Regression Model

- Model has an R-squared of 0.01
- Try it out anyway backtesting since Feb 1 2021
- Interpret model as setting a buy signal if model predicts $\geq 1\%$ growth for next candle and sell signal otherwise
- Mostly ends up buying during both run-ups and corrections with a few sells

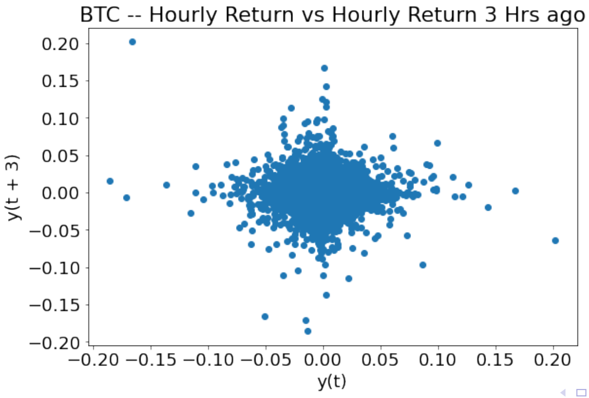


Regression Model Conclusions

- Both dependent and independent variables are arbitrarily defined
- No relationships between the independent and dependent variable
- Need a better defined outcome variable
- Need better defined predictors, maybe a focus on on/off indicators instead of continuous quantites
- Probably want to supplement with non-price data, such as on-chain or social data

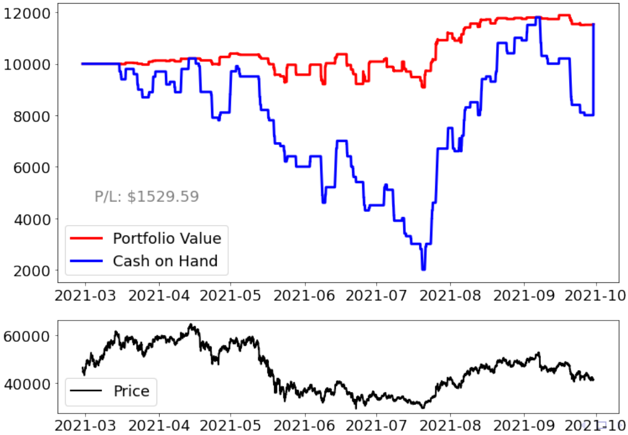
Other Modeling Approaches

- Random Forest - Using Buy/Sell Signal from 5% up/down first calculated quantity
- Similarly poor behavior (no predictive value) when data is split train/test by date
- ARIMA - Extrapolate based on close data, would require correlations between current and prior return data



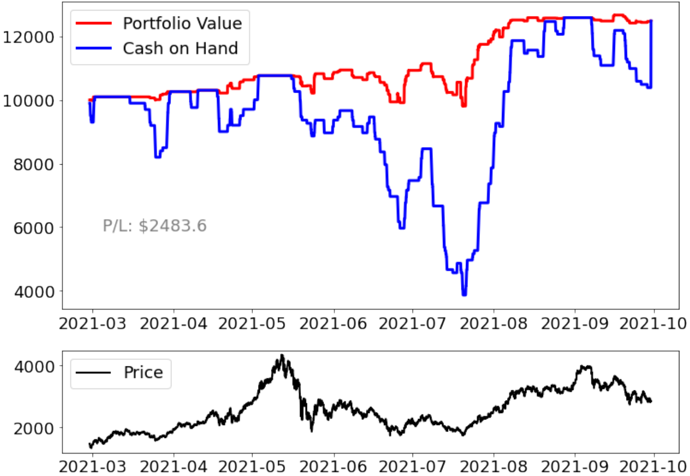
Better Signals

- Try a simple RSI/Moving Average based reversal signal
- Sell if RSI $>$ 70 and uptrend (Close $>$ SMA-50)
- Buy if RSI $<$ 30 and downtrend (Close $<$ SMA-50)



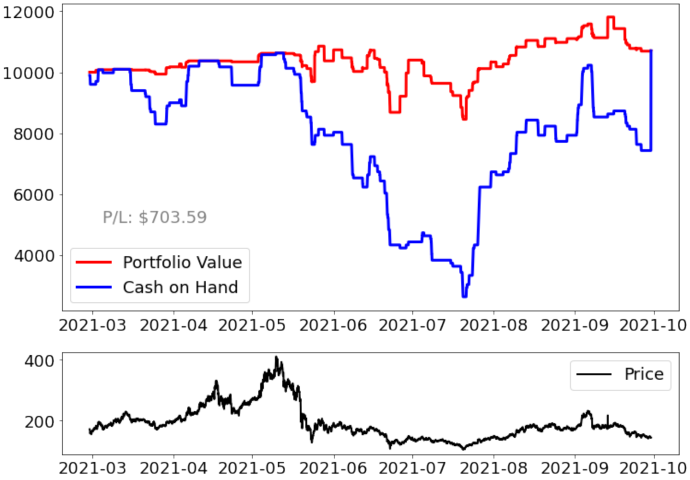
RSI Signal - ETH 1H

- Does well on 1H ETH



RSI Signal - LTC 1H

- As well as 1H LTC



Future Directions

- Focus on improving Signal Outcome Column and Calculated Indicators
- Add non price based data, such as on-chain, social, fundamentals
- Check Academic Literature: Sebastiao and Godinho 2021 Financial Innovation
- Focus more on indicators as opposed to continuous quantities
- Focus on utilizing improvements to default Random Forest/Regression tools

RESEARCH Open Access

Forecasting and trading cryptocurrencies with machine learning under changing market conditions

Helder Sebastião* and Pedro Godinho

*Correspondence: helderse@fe.uc.pt
Univ Coimbra, CeBER, Faculty of Economics, Av. Dr. Dias da Silva, 165, 3004-512 Coimbra, Portugal

Abstract

This study examines the predictability of three major cryptocurrencies—bitcoin, ethereum, and litecoin—and the profitability of trading strategies devised upon machine learning techniques (e.g., linear models, random forests, and support vector machines). The models are validated in a period characterized by unprecedented turmoil and tested in a period of bear markets, allowing the assessment of whether the predictions are good even when the market direction changes between the validation and test periods. The classification and regression methods use attributes from trading and network activity for the period from August 15, 2015 to March 03, 2019, with the test sample beginning on April 13, 2018. For the test period, five out of 18 indi-

Bonus Slide – Backtesting

```python
# Provide for backtesting
class Strategy:
    def __init__(self, history=1):
        self.historyNeeded = history

    def apply(self, df, ledger=None):
        pass

class Backtest:
    def __init__(self, priceDataFrame, initialValue=10000, unitPrice=100, tradeFeePct=0.0, logging=True):
        self.strategies = {}
        self.priceData = priceDataFrame
        self.initialValue = initialValue
        self.value = initialValue
        self.unitPrice = unitPrice
        self.amount = 0.0
        self.tradingFeePct = tradeFeePct/100
        self.startDate = self.priceData.Date.min()
        self.endDate = self.priceData.Date.max()

        # Turn ledger off if needed
        self.ledger = []
        self.log = logging

```

Bonus Slide – Backtesting

```
    def evaluateStrategy(self, strategy, name=None, passLedger=False):
        for index, row in self.priceData.iterrows():
            #print ('BEGIN STRATEGY LOOP')
            #print (index, row['Date'], row['Open'], row['Close'], row['RSI-14'])
            strategyFrame = self.priceData.iloc[index-strategy.historyNeeded:index]
            if (len(strategyFrame) == strategy.historyNeeded):
                if (passLedger):
                    if (name):
                        #print ('HERE!')
                        signal, size = strategy.apply(self.priceData.iloc[index+1-strategy.historyNeeded:index+1]
                                                ledger=self.ledger, unitPrice=self.unitPrice)
                    else:
                        signal, size = strategy.apply(self.priceData.iloc[index+1-strategy.historyNeeded:index+1]
                                                ledger=self.ledger, unitPrice=self.unitPrice)
                else:
                    signal, size = strategy.apply(self.priceData.iloc[index+1-strategy.historyNeeded:index+1],
                                            ledger=None, unitPrice=self.unitPrice)

                if (signal == 'buy'):
                    #print (size, row['Close'], self.unitPrice, size*row['Close']/self.unitPrice)
                    self.buyAsset(size*row['Close']/self.unitPrice, row['Close'], row['Date'])
                elif (signal == 'sell'):
                    self.sellAsset(size*row['Close']/self.unitPrice, row['Close'], row['Date'])

            #print ('END STRATEGY LOOP')
```

Bonus Slide – Backtesting

```python
def pltStrategy(self, name):
    logDF = pd.DataFrame(self.strategies[name]['log'])
    #logDF = logDF.merge(self.priceData[['Date','Close']])
    logDF = self.priceData[['Date','Close']].merge(logDF, how='outer')
    logDF = logDF.fillna(method='ffill').fillna(method='bfill')

    fig, axs = plt.subplots(2,1, gridspec_kw={'height_ratios': [3, 1]})
    axs[0].plot(logDF['Date'],logDF['PortfolioValue'], color='red', label='Portfolio Value', linewidth=3)
    axs[0].plot(logDF['Date'],logDF['PortfolioCash'], color='blue', label='Cash on Hand', linewidth=3)
    axs[0].legend()
    axs[1].plot(logDF['Date'],logDF['Close'], color='k', label='Price', linewidth=2)
    axs[1].legend()
    plt.show()
```

Bonus Slide – RSI

```python
class RSI(Strategy):
    def apply(self, df, ledger=None, unitPrice=100.0):
        # RSI Signals
        if ((df.iloc[0]['RSI-14'] < 30.0) and (df.iloc[0]['Close'] < df.iloc[0]['SMA-50'])):
            return 'buy', unitPrice/df.iloc[0]['Close']
        elif ((df.iloc[0]['RSI-14'] > 70.0) and (df.iloc[0]['Close'] > df.iloc[0]['SMA-50'])):
            return 'sell', unitPrice/df.iloc[0]['Close']
        else:
            return 'none', 0

```

Bonus Slide – Regression

```python
class RegressionModel(Strategy):
    def apply(self, df, ledger=None, unitPrice=100.0):
        ### -0.0079 + 0.0000007548*Volume + 0.0002*RSI-14 - 0.0611*EMA-20-deviation +
        ###  0.00001388dSMA-5dt -0.0520logReturn
        predictedReturn = -0.0079
        predictedReturn += 0.0000007548*df.iloc[0]['Volume']
        predictedReturn += 0.0002*df.iloc[0]['RSI-14']
        predictedReturn -= 0.0611*df.iloc[0]['EMA-20-deviation']
        predictedReturn += 0.00001388*df.iloc[0]['dSMA-5dt']
        predictedReturn -= 0.0520*df.iloc[0]['logReturn']

        if (predictedReturn >= 0.01):
            return 'buy', unitPrice/df.iloc[0]['Close']
        elif (predictedReturn <= -0.01):
            return 'sell', unitPrice/df.iloc[0]['Close']
        else:
            return 'none', 0
```

Bonus Slide – Collapse

```python
    # Create longer timeframe candles
    # FIXME: There is definitely a more pythonic way to do this...
    def collapse(self, window):
        descOHLC = self.df.sort_values(by=['Date'], ascending=False, ignore_index=True)
        collapsedOHLC = []
        index = 0
        while (index+window-1 <= descOHLC.index[-1]):
            collapseRow = {}
            collapseRow['Unix Timestamp'] = descOHLC.iloc[index+window-1]['Unix Timestamp']
            collapseRow['Date'] = descOHLC.iloc[index+window-1]['Date']
            collapseRow['Symbol'] = descOHLC.iloc[index]['Symbol']
            collapseRow['Open'] = descOHLC.iloc[index+window-1]['Open']
            collapseRow['High'] = descOHLC.iloc[index:index+window]['High'].max()
            collapseRow['Low'] = descOHLC.iloc[index:index+window]['Low'].min()
            collapseRow['Close'] = descOHLC.iloc[index]['Close']
            collapseRow['Volume'] = descOHLC.iloc[index:index+window]['Volume'].sum()
            collapsedOHLC.append(collapseRow)
            index = index + window

        return pd.DataFrame(collapsedOHLC)
```

Other Indicators

```python
    # Add simple returns calculation
    def addArithmeticReturns(self):
        columnName = 'Return'
        self.df[columnName] = (self.df['Close'] - self.df['Close'].shift(1))/self.df['Close'].shift(1)

    # Add logarithmic returns calculation (https://quantivity.wordpress.com/2011/02/21/why-log-returns/)
    def addLogReturns(self):
        columnName = 'logReturn'
        self.df[columnName] = np.log(self.df['Close'] / self.df['Close'].shift(1))

    # Add Relative Price Range (Used in Sebastiao and Godinho 2021)
    def addRRt(self):
        columnName = 'RRt'
        self.df[columnName] = 2.0*(self.df['High'] - self.df['Low']) / (self.df['High'] + self.df['Low'])

    # Add Parkinson Volatility Estimator (Used in Sebastiao and Godinho 2021 from Parkinson 1980)
    def addParkVol(self):
        columnName = 'sigmat'
        self.df[columnName] = np.sqrt((np.log(self.df['High']/self.df['Low'])**2)/(4.0 * np.log(2)))

    # Add Simple Moving Average Indicator for desired timeframe
    def addSMA(self, window):
        columnName = "SMA-" + str(window)
        self.df[columnName] = SMAIndicator(close=self.df['Close'], window=window, fillna=False).sma_indicator()

        columnName2 = "dSMA-" + str(window) + "dt"
        self.df[columnName2] = (self.df[columnName] - self.df[columnName].shift(1))
```