Udacity Machine Learning Engineer Nanodegree

Cryptocurrency Signal Generation Capstone Project

Ronen Shohat

1. Definition

Description/Overview

Asset managers have been utilizing algorithmic models to generate investment decisions for decades, using both real-time and historical data. With the introduction of the first cryptocurrency, Bitcoin, in 2009 along with its emergence in popularity in 2017 as major cryptocurrencies hit all-time highs, the growth of crypto asset managers has emerged as a new industry. These firms are trying to implement similar quantitative investment strategies found in traditional asset classes to build predictive models for cryptocurrency prices.

A few academic studies related to tracking cryptocurrency prices can be found here:

https://scholar.smu.edu/cgi/viewcontent.cgi?article=1039&context=datasciencereview

https://pdfs.semanticscholar.org/477c/211d4bcb3668906513e31aba951da1b33128.pdf

Problem Statement

Although the technology of cryptocurrencies has substantial potential, its volatile price swings have deterred many investors from entering the market and has seen other investors suffer substantial losses. In order to solve this issue, I will try to determine key features that drive the price of BTC to gain insight on the forward direction of price, in order to generate trading strategies (which will be further discussed).

Datasets and Inputs

CryptoCompare API- price and volume data for BTC/USD prices as well as other major cryptocurrencies (LTC, ETH, XRP, BCH, XMR)

Blockchain website for data related to bitcoin market structure/fundamentals

Label will be based on the week over week percentage price of BTC/USD. There will be 3 classes:

- Downward (DoD change was less than -1%)
- Stable (DoD change was between -1% and 1%)

- Upward (DoD change was greater than 1%)

There are 100 features (including transformations) in the dataset, with 558 daily data points ranging from 2017-08-30 to 2019-03-10

The list of features are:

['close',

'high',

'low',

'open',

'volumefrom',

'volumeto',

'intraday_range',

'dod_change',

'rolling_price_vol_7',

'rolling_price_vol_30',

'rolling_DoD_vol_7',

'rolling_DoD_vol_30',

'close_ETH',

'high_ETH',

'low_ETH',

'open_ETH',

'volumefrom_ETH',

'volumeto_ETH',

'intraday_range_ETH',

'dod_change_ETH',

'rolling_price_vol_7_ETH',

'rolling_price_vol_30_ETH',

'rolling_DoD_vol_7_ETH',

'rolling_DoD_vol_30_ETH',

'close_LTC',

'high_LTC',

'low_LTC',

'open_LTC',

'volumefrom_LTC',

'volumeto_LTC',

'intraday_range_LTC',

'dod_change_LTC',

'rolling_price_vol_7_LTC',

'rolling_price_vol_30_LTC',

'rolling_DoD_vol_7_LTC',

'rolling_DoD_vol_30_LTC',

'close_XRP',

'high_XRP',

'low_XRP',

'open_XRP',

'volumefrom_XRP',

'volumeto_XRP',

'intraday_range_XRP',

'dod_change_XRP',

'rolling_price_vol_7_XRP',

'rolling_price_vol_30_XRP',

'rolling_DoD_vol_7_XRP',

'rolling_DoD_vol_30_XRP',

'close_BCH',

'high_BCH',

'low_BCH',

'open_BCH',

'volumefrom_BCH',

'volumeto_BCH',

'intraday_range_BCH',

'dod_change_BCH',

'rolling_price_vol_7_BCH',

'rolling_price_vol_30_BCH',

'rolling_DoD_vol_7_BCH',

'rolling_DoD_vol_30_BCH',

'close_XMR',

'high_XMR',

'low_XMR',

'open_XMR',

'volumefrom_XMR',

'volumeto_XMR',

'intraday_range_XMR',

'dod_change_XMR',

'rolling_price_vol_7_XMR',

'rolling_price_vol_30_XMR',

'rolling_DoD_vol_7_XMR',

'rolling_DoD_vol_30_XMR',

'block_size',

'cost_pct_volume',

'cost_per_transaction',

'mining_difficulty',

'transaction_volume_usd',

'transaction_volume',

'tera_hashes',

'confirm_time',

'mempool_count',

'mempool_size',

'mining_revenue',

'sent_btcusd',

'sent_buybitcoin',

'sent_bitcoin',

'sent_cryptocurrency',

'wallet_users',

'trans_ex_long',

'trans_ex_popular',

'transactions_per_block',

'transactions_total',

'trans_per_day',

'unique_addresses',

'output_volume',

'num_bitcoins',

'usd_volume',

'mining_fees_usd',

'trans_per_second',

'unspent_transactions']

Solution Statement

This project is going to try to build predictive signals on whether a specific cryptocurrency will increase in price, decrease in price, or remain stable. As a classification problem, we will test several supervised learning algorithms to optimize predictive performance.

Potential Algorithms to use:

- Random Forest
- Gradient Boosting
- XGBoost
- SVM (RBF and sigmoid kernels)
- Decision Trees

An ensemble algorithm would be a preferred method as global and local interpretation of the results are very transparent in determining which features drove the prediction

Metrics: Evaluation Metrics (Absolute Alpha vs Relative performance vs benchmark)

Apart from traditional metrics like precision and recall, the main measure that will be used is the USD gained from implementing the algorithm's price predictions, after considering trading costs. We will also assess traditional performance metrics like precision and recall, while considering the fact that 1 step misclassifications (ie upward vs stable) should be penalized less than 2 step classifications (ie downward vs upward). This will determine whether this algorithm provides better price prediction than simply buying and holding the asset.

Formulas for Precision and Recall:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Project Design

- Collecting/Structuring Data from various APIs
- Data Preprocessing/Cleaning
- Feature engineering
- Model testing/training

- Deciding between regression and classification
- Picking optimal model based on continuous P&L backtests
- Model tuning with gridsearch
- Final conclusion

2. Analysis

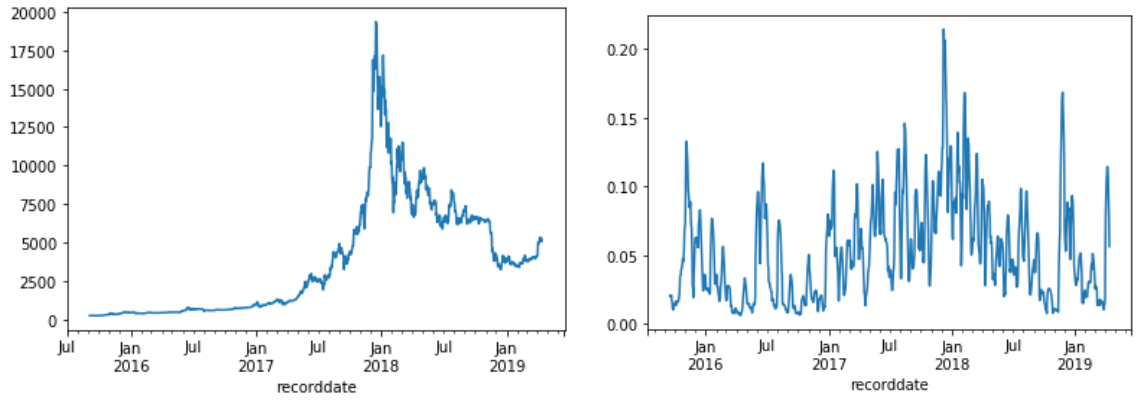
Feature Engineering

In order to capture changes in momentum over time, I decided to add rolling percent changes and standard deviations over a DoD, WoW, and MoM period

```
df['rolling_price_vol_7'] =  df['open'].rolling(7).std() / df['open'].rolling(7).mean()
df['rolling_price_vol_30'] =  df['open'].rolling(30).std() / df['open'].rolling(30).mean()
df['rolling_DoD_vol_7'] =  df['dod_change'].rolling(7).std() / df['dod_change'].rolling(7).mean()
df['rolling_DoD_vol_30'] =  df['dod_change'].rolling(30).std() / df['dod_change'].rolling(30).mean()
```

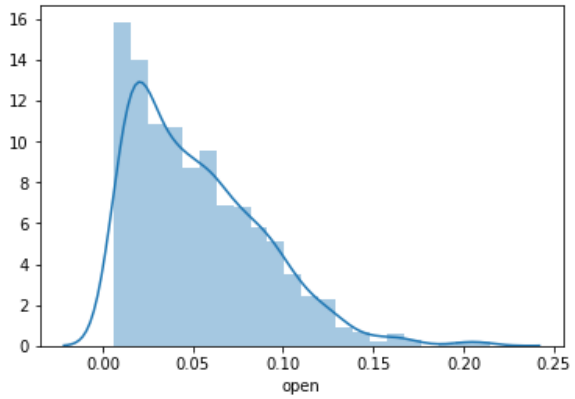
Data Exploration/Exploratory Visualization

Visually, it is apparent that cryptocurrencies have experienced incredible volatility over time, peaking at almost $20,000 at the end of 2017. The asset has also experienced significant levels of volatility over time, with mean adjusted standard deviation hovering at 0.1. There is also significant tail risk in the distribution of volatility over time



```
sns.distplot((master_df['open'].rolling(14).std() / master_df['open'].rolling(14).mean()).dropna())
```

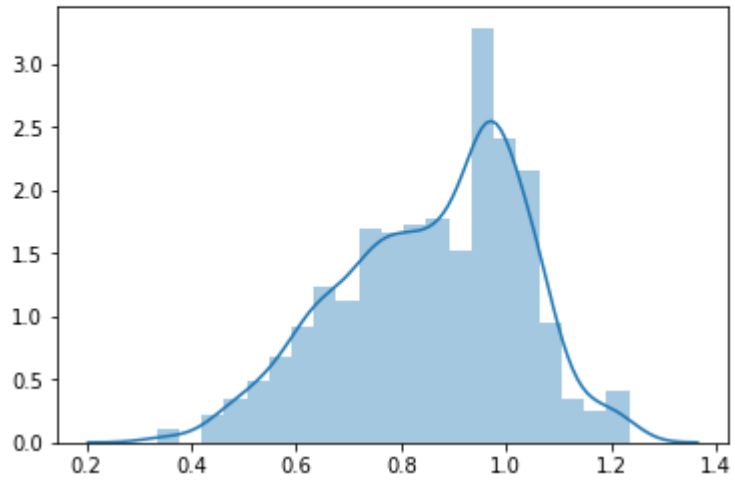
```
<matplotlib.axes._subplots.AxesSubplot at 0x26b27b6ac18>
```



In terms of other fundamental data, there were few outliers in the set and all features showed relatively normal distributions

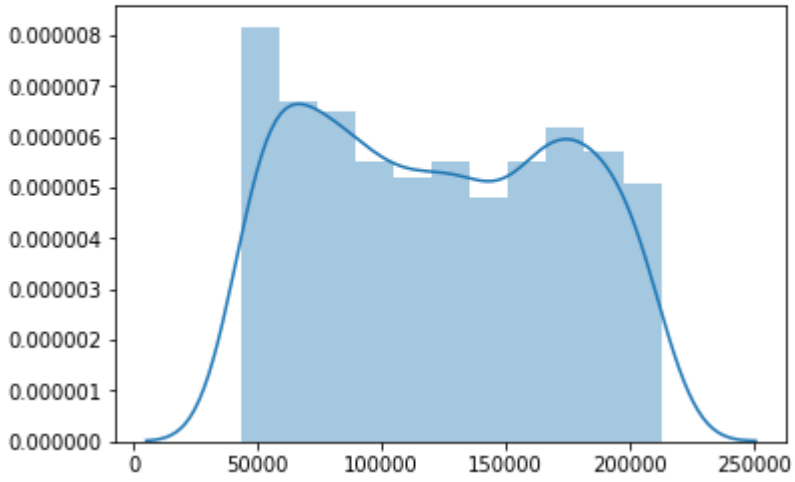
```
sns.distplot(master_df[['avg-block-size']])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x26b28a6
```



```
sns.distplot(master_df[['blocks-size']])
```

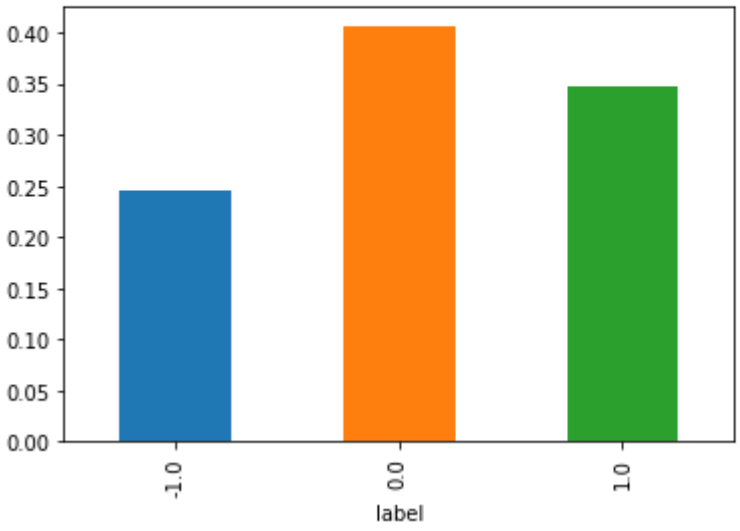
```
<matplotlib.axes._subplots.AxesSubplot at 0x26b28afbc18>
```



In terms of the distribution of the labels, there wasn't an incredibly significant imbalance, though labels weren't perfectly balanced

```
(master_df.groupby('label').size() / len(master_df)).plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x26b2a165940>
```



Algorithms and Techniques

To solve this problem, I intend on utilizing supervised techniques to build a classification model, where the label is the t+3 price direction of BTC. There will be 3 possible values for the label, -1 if down, 0 if stable, 1 if up. Out of the several classification models I tried, ensemble methods seemed to be the most effective in terms of speed and performance. In this study I will explore the results of popular bagging method Random Forest and another popular boosting method, XGBoost. Both algorithms are widely used in the industry and are well known for their impressive performance, fast learning speed (compared to SVC for example), and model interpretability (ie local and global feature importances).

Metrics/Analysis: Benchmark Model

To benchmark our model performance, we will backtest the algo's investment decisions to calculate a P&L amount in USD, then compare to a passive strategy of just buy-and-hold.

3. Methodology

Preprocessing/Time Series Matching

Since I am trying to model predictions in price movement, I had to make sure my forward looking model wasn't incorporating data from the future to predict the past. I also had to adjust my data to make sure it would be available within the same time frame to make my predictions. Because of this, I decided to model a prediction in price change t+2, and was thus unable to incorporate any data in the model that was available later than that

```
t = 2
t = -t - 1

master_df = pd.merge(prices_df, excels_df, on = ['recorddate'], how = 'outer').sort_values('recorddate')
master_df = master_df.set_index('recorddate')

t_list = []
for col in master_df.columns:
    if float(master_df.loc[master_df.index == list(excels_df['recorddate'])[t+1 ], col].isna()) == 0:
        t_list.append(col)
master_df = master_df[t_list]
```

```
master_df = master_df.drop(columns = [s for s in master_df.columns if "close" in s])
master_df[[s for s in master_df.columns if s in ('high','low','intraday_range','volumefrom','volumeto')]] = master_df[[s for s i
master_df[[s for s in master_df.columns if s in ('dod_change')]] = master_df[[s for s in master_df.columns if s in  ('dod_change

master_df['pred_dod_change'] = master_df['dod_change'].shift(t)

master_df = master_df.dropna()
master_df=master_df.replace([np.inf,-np.inf],0).fillna(0)
```

Model Labeling

I created the shifted label pred_dod_change to generate a prediction t-3 days in advance, so that I could make an actionable trade at t+2. I then bucketed the changes into 3 buckets; -1 if below -0.01, 0 if between 0 and 0.01, and 1 if greater than 0.01. The distributions weren't perfectly even at approximately 25/40/35, but it wasn't extreme enough to warrant a resampling of the data

```
bins = [-np.inf,-0.01,0.01,np.inf]
buckets = ['-1','0','1']

master_df['label'] = pd.cut(master_df['pred_dod_change'], bins = bins, labels = buckets).astype(float)
#master_df['label'] = np.where(master_df['pred_dod_change'] > 0.01, 1, 0)

drop_list = ['pred_dod_change','label','dod_change']
X = master_df.drop(columns = drop_list)
y = master_df['label']

X = X.astype(float)
split_cutoff_date = '2018-06-01'

X_train, X_test = X.loc[(X.index < split_cutoff_date)], X.loc[(X.index >= split_cutoff_date) ]
y_train, y_test = y.loc[ (y.index < split_cutoff_date)], y.loc[(y.index >= split_cutoff_date) ]
```

```
master_df.groupby('label').size() / len(master_df)
```

```
label
-1.0    0.246388
 0.0    0.406084
 1.0    0.347529
dtype: float64
```

Initial Model Creation

Based on the shifted labels, I fit Random Forest and XGBoost using the training data and then analyzed performance on the test dataset. I also made sure to drop any features that aren't available in the past like dod_change or pred_dod_change, as that would make the model unusable in a real-life setting.

```
bins = [-np.inf,-0.01,0.01,np.inf]
buckets = ['-1','0','1']

master_df['label'] = pd.cut(master_df['pred_dod_change'], bins = bins, labels = buckets).astype(float)
#master_df['label'] = np.where(master_df['pred_dod_change'] > 0.01, 1, 0)

drop_list = ['pred_dod_change','label','dod_change']
X = master_df.drop(columns = drop_list)
y = master_df['label']

X = X.astype(float)
split_cutoff_date = '2018-06-01'

X_train, X_test = X.loc[(X.index < split_cutoff_date)], X.loc[(X.index >= split_cutoff_date) ]
y_train, y_test = y.loc[ (y.index < split_cutoff_date)], y.loc[(y.index >= split_cutoff_date) ]
```

```
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

rf_model_opt = RandomForestClassifier(
                                    max_depth = 24, min_samples_split = 7, min_samples_leaf = 0.1)
rf_model_opt.fit(X_train, y_train)

model_clss = XGBClassifier()
model_clss.fit(X_train, y_train)
model_clss_opt = XGBClassifier(max_depth = 6, min_child_weight = 3, gamma = 0, colsample_bytree =0.9, learning_rate = 0.3)
model_clss_opt.fit(X_train, y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.9, gamma=0, learning_rate=0.3, max_delta_step=0,
       max_depth=6, min_child_weight=3, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='multi:softprob', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

Refinement: Model Selection/Hyperparameter Optimization

After experimenting with several classification algorithms, it seemed ensemble methods yielded the best performance. I decided to fit Random Forest and XGBoost models on my labels, as well as created versions of these models with optimized hyperparameters (using a time series validated Grid Search). Due to the nature of this problem, I emphasized performance as the precision of label 1. Because most exchanges don't allow short selling, having high precision for 1 would allow me to decide whether to buy if there is an upside opportunity.

```
from sklearn.metrics import make_scorer
def precision_up(y_true, y_pred):
    fscore_list = precision_recall_fscore_support(y_true,y_pred)[2]
    precision_list = precision_recall_fscore_support(y_true,y_pred)[0]
    if np.where(fscore_list==0)[0].size > 0:
        output = 0
    else:
        output = precision_list[1]
    return output


scorer = make_scorer(precision_up, greater_is_better=True)
```

```
param_test1 = {
 'max_depth':np.linspace(1, 50, 50, endpoint=True),
    'min_samples_split': range(2,15),
    'min_samples_leaf': np.linspace(0.1, 0.5, 9, endpoint=True)
}


gsearch1 = GridSearchCV(estimator = RandomForestClassifier( ), scoring = scorer,
 param_grid = param_test1, n_jobs=-1,iid=False, cv=cv)
gsearch1.fit(X_train, y_train)
print(gsearch1.best_params_, gsearch1.best_score_)
gs_output(gsearch1)
```

```
{'max_depth': 24.0, 'min_samples_leaf': 0.1, 'min_samples_split': 7} 0.20757575757575755
```

	index	max_depth	min_samples_leaf	min_samples_split
2696	0.207576	24.0	0.10	7
1177	0.205797	11.0	0.10	9

```
param_test7 = {
 'learning_rate':[0.001,0.01,0.1,0.17,0.2,0.3,0.7,0.8,0.9,1],
    'reg_alpha' : [0,1e-7,1e-5,0.001]
}
gsearch7 = GridSearchCV(estimator = XGBClassifier(max_depth = 6, min_child_weight = 3, gamma = 0,
                                                  colsample_bytree = 1, substample = 1),
 param_grid = param_test7, scoring= scorer,n_jobs=-1,iid=False, cv=cv)
gsearch7.fit(X_train, y_train)
print(gsearch7.best_params_, gsearch7.best_score_)
gs_output(gsearch7)
```

```
{'learning_rate': 0.3, 'reg_alpha': 0} 0.4214135386218001
```

	index	learning_rate	reg_alpha
20	0.421414	0.300	0.000000e+00
21	0.421414	0.300	1.000000e-07

4. Results

Performance Analysis

Random Forest Results:

```
print(classification_report(y_test,y_pred_rf))
print('Hyperparam optimized: \n')
print(classification_report(y_test,y_pred_rf_opt))
```

```
              precision    recall  f1-score   support

        -1.0       0.40      0.17      0.24        83
         0.0       0.53      0.17      0.26       144
         1.0       0.27      0.71      0.39        87

   micro avg       0.32      0.32      0.32       314
   macro avg       0.40      0.35      0.30       314
weighted avg       0.42      0.32      0.29       314

Hyperparam optimized:

              precision    recall  f1-score   support

        -1.0       0.00      0.00      0.00        83
         0.0       0.66      0.31      0.42       144
         1.0       0.31      0.87      0.46        87

   micro avg       0.39      0.39      0.39       314
   macro avg       0.32      0.40      0.29       314
weighted avg       0.39      0.39      0.32       314
```

XGBoost Results:

	precision	recall	f1-score	support
-1.0	0.27	0.05	0.08	83
0.0	0.65	0.24	0.35	144
1.0	0.29	0.82	0.43	87
micro avg	0.35	0.35	0.35	314
macro avg	0.40	0.37	0.29	314
weighted avg	0.45	0.35	0.30	314

Hyperparam optimized:

	precision	recall	f1-score	support
-1.0	0.23	0.07	0.11	83
0.0	0.57	0.32	0.41	144
1.0	0.29	0.70	0.41	87
micro avg	0.36	0.36	0.36	314
macro avg	0.37	0.36	0.31	314
weighted avg	0.41	0.36	0.33	314

As noted in the results, the hyperparameter tuned Random Forest seemed to yield the best performance in terms of label 1 precision. However, at 0.31 it seemed unlikely this would yield significant alpha generating opportunities.

Feature Importances of Optimized Random Forest

```
pd.DataFrame(rf_model_opt.feature_importances_, index=X.columns,
                          columns=["Importance"]).sort_values('Importance', ascending = False)
```

	Importance
intraday_range_pct	0.152674
intraday_range	0.147201
volumeto	0.112908
volumefrom	0.109232
trade-volume	0.079561
rolling_price_vol_30	0.066218
open_ETH	0.061609
volumeto_LTC	0.056223
high_XRP	0.030406

Based on feature importances, it seems most weight was put towards trade volume and volatility metrics (intraday range, rolling 30 days standard deviation)

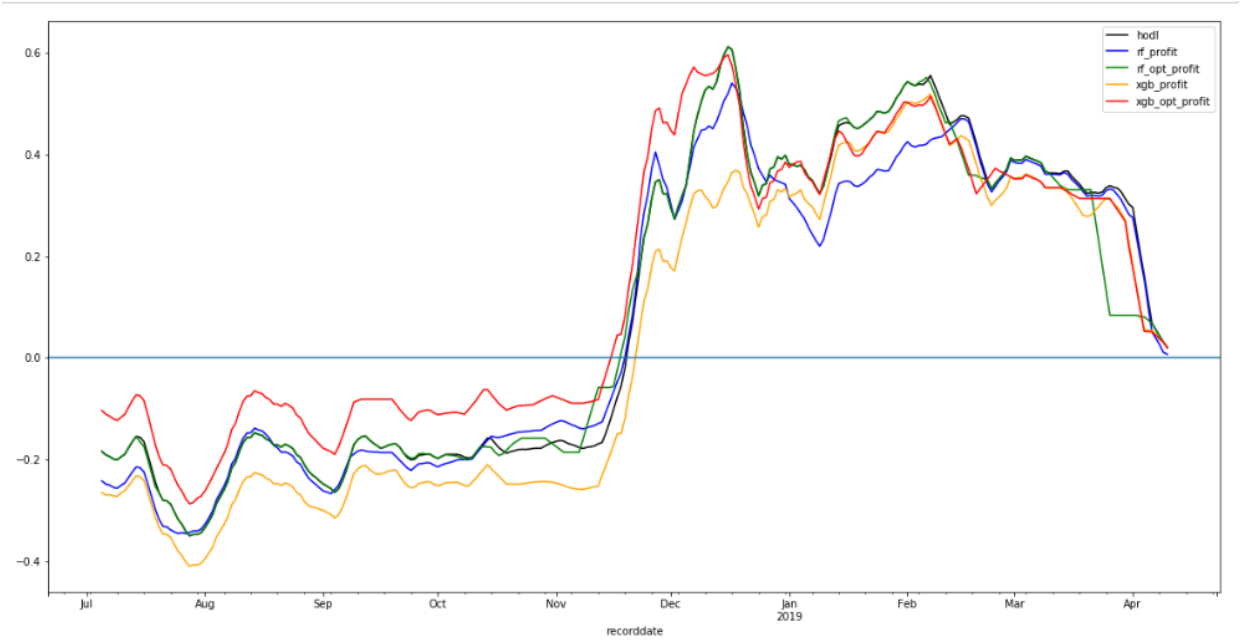
Backtesting Model Evaluation

Due to the lack of any strong overall performance, I tried experimenting with trading strategies that would rely more on label 1 precision. Because of this I designed 3 trading strategies:

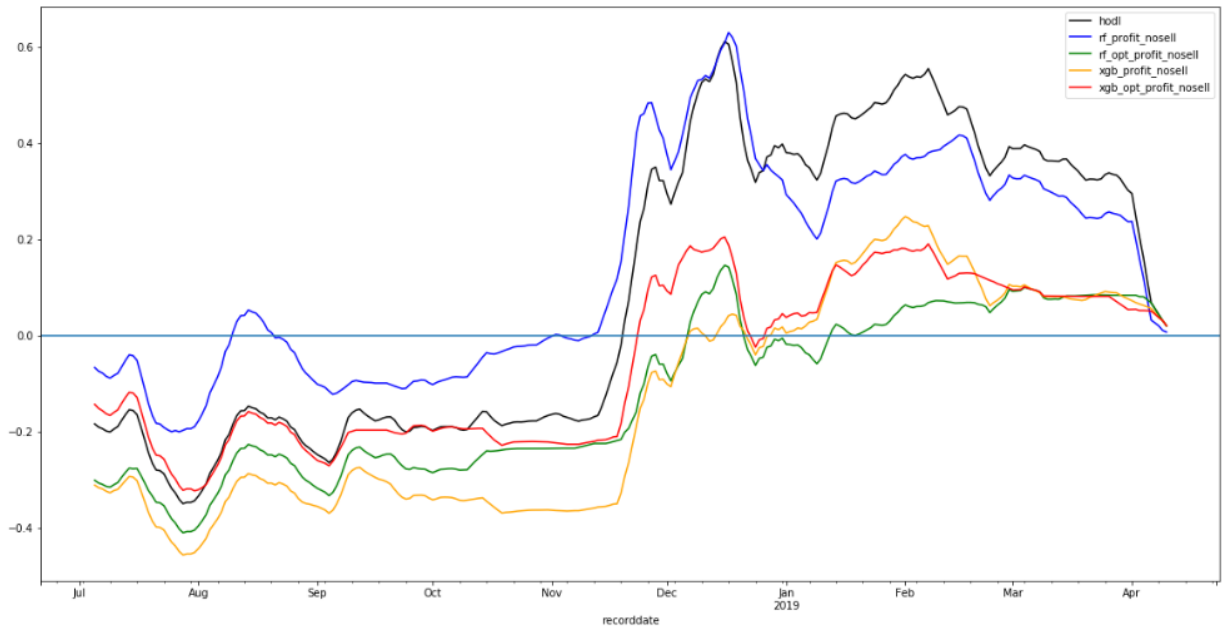
1. Profit: Buy if label is 1 and you have USD. Sell if label is -1 and you have BTC
2. Profit_nosell: Buy if label is 1 and you have USD. Sell if label is != 1 and you have BTC
3. Profit_nosell_churn: Buy if label is 1 and you have USD. Then sell your BTC position next day into USD

I then decided to compare both absolute and relative returns (vs just holding BTC **shown in the black line**) over the testing time period.

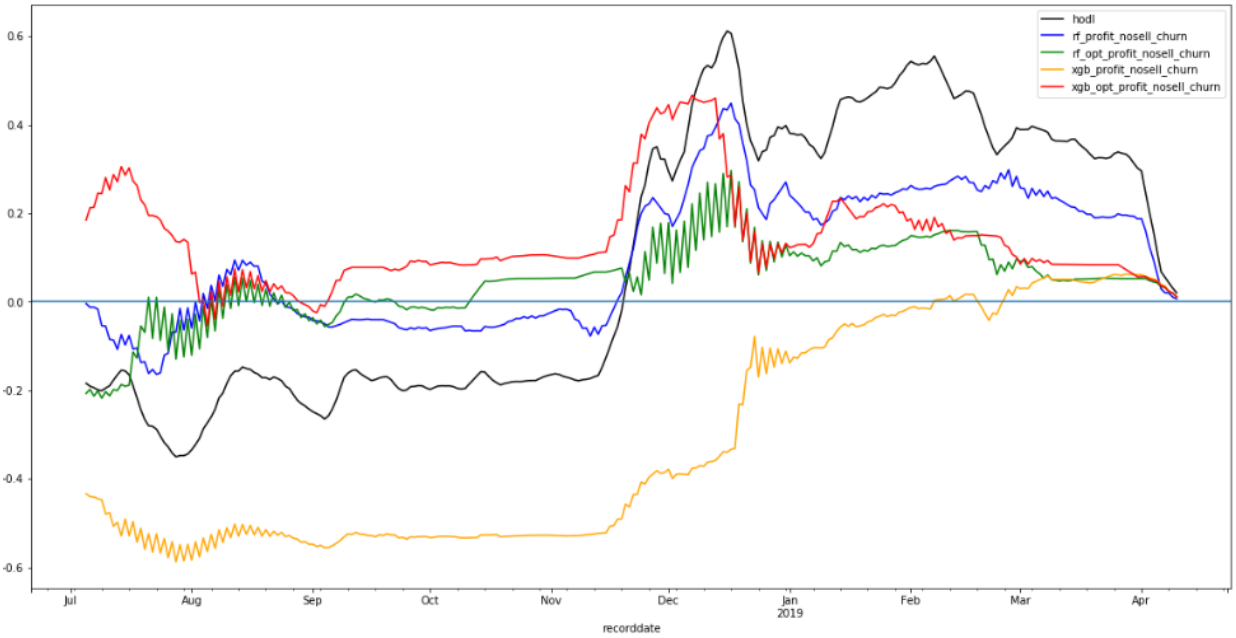
Performance of Strategy 1:



Performance of Strategy 2:



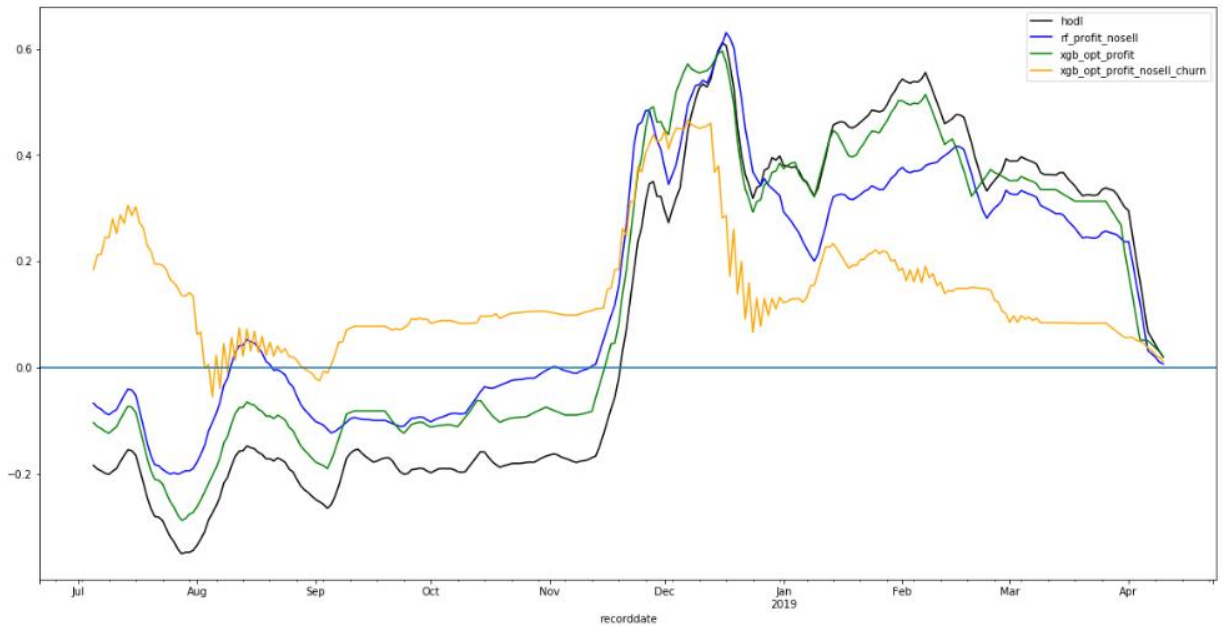
Performance of Strategy 3:



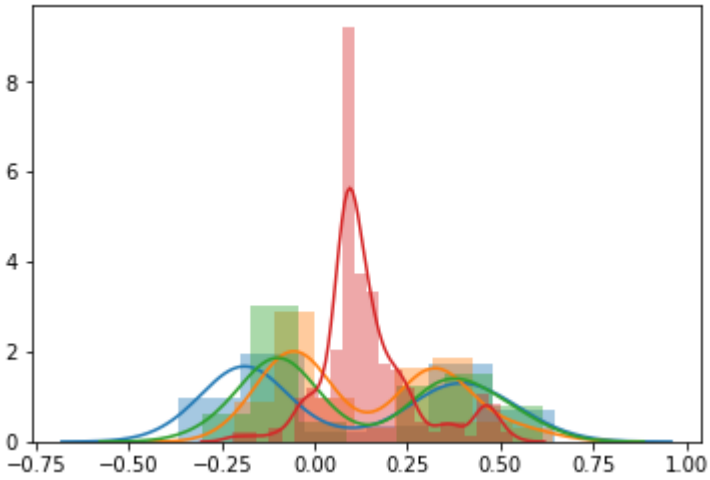
5. Conclusion

Free Form Visualization

Performance of strategies compared:



Distribution of annualized returns over time:



Reflection

Based on the backtesting, there didn't seem to be any specific strategy that significantly outperformed on a relative basis when compared to just holding BTC. However, the churn strategy using the XGB optimized model seemed to have the most impressive absolute return stability when compared to any other strategy. When backtested, it seemed to consistently yield positive returns with much lower standard deviation than other strategies

```
(scen_df.sub(scen_df['hodl'], axis = 0).mean() / scen_df.std()).sort_values(ascending = False)
```

```
xgb_opt_profit_nosell_churn    0.360218
rf_profit_nosell               0.193805
xgb_opt_profit                 0.142553
rf_profit_nosell_churn         0.024370
hodl                           0.000000
rf_opt_profit                 -0.027698
rf_profit                     -0.072617
xgb_profit                    -0.218319
rf_opt_profit_nosell_churn    -0.330608
xgb_opt_profit_nosell         -0.895663
xgb_profit_nosell             -1.044667
rf_opt_profit_nosell          -1.290417
xgb_profit_nosell_churn       -1.588883
dtype: float64
```

In this sense, the model did seem to be successful in creating an alpha opportunity given an absolute return strategy, and delivering significant risk adjusted return. Yielding a Sharpe ratio of 0.36 isn't significantly impressive, but is respectable

Improvement

In terms of improvement of the model, I feel next steps are to gather more insightful features from other external sources and invest more time in engineering features. For example, gathering and cleaning data related to real-time sentiment on crypto forums or social media platforms could potentially have significant explanatory power. Another source of improvement could be on the model generation end. While I have been focusing on ensemble methods in this project, I could experiment with implementing recurrent neural networks and evaluate performance. For example, LSTMs (Long Short Term Memory) are renowned for recognizing both longer and short term patterns in data. Having the ability to create a robust model that captures both long and short term trends could yield higher performance, and ultimately higher outperformance.