Introduction

This project is about creating a model that trains itself of market data and creates a gradient 'boosting' decision tree algorithm to create buy/sell signals based upon return forecasts. The idea behind 'boosting' is that you train base learners (which in this case they are decision trees) into an ensemble. The idea behind boosting is to combine the predictions of weak models (only slightly outperforming a random guess) and output an ensemble model with higher accuracy. The goal of boosting is to ensure that the newest base learner compensates for the shortcomings of the previous learner. Each addition directly contributes to reducing the overall training error, given the errors made by prior ensemble members. The end prediction is a weighted average from all the previous models. Boosting has often demonstrated remarkable resilience to overfitting, especially when combined with shrinkage techniques (which the LightGBM implementation uses) despite significant growth of the ensemble and, thus, the complexity of the model. This is the reason it is used for this test case, since overfitting for noise is a worry with training a model on market data.

The gradient aspect of the model is alluding to the idea that the algorithm is training itself off of the negative gradient of the loss function of the ensemble. Each step will reduce the training error.

The model in this example fits the learners (h_m), or the decisions trees, to the gradient of the loss function, which is the following formula:

$$H_m(x) = \underbrace{H_{m-1}(x)}_{\text{current ensemble}} + \underbrace{\gamma_m h_m(x)}_{\text{new member}} = H_{m-1}(x) + \operatorname*{argmin}_{\gamma,h} \sum_{i=1}^{N} \underbrace{L(y_i, H_{m-1}(x_i) + h(x))}_{\text{loss function}}$$

The algorithm calculates the gradient for each observation, then fits a regression tree to the residuals which are created. It also creates a prediction for each leaf node that will minimize the incremental loss due to adding the new learner to the ensemble.

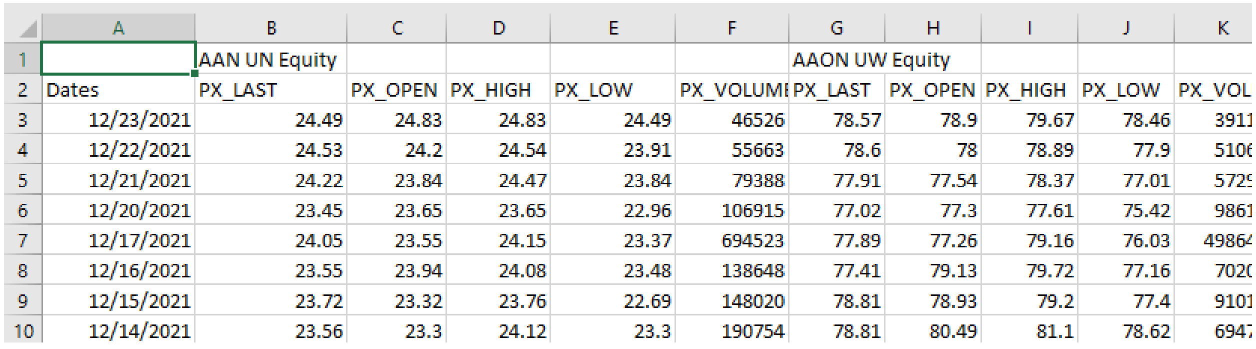
The final model makes predictions based on the weighted sum of the predictions created by all the individual learners. This model learns both classification and regression rules.

The ML hyperparameters are chosen by cross-validating several different configurations for different amounts of boosting, lookahead windows, as well as the roll-forward periods and other hyperparameter adjustments.

Data Ingestion

The data used in this model is the adjusted open, close, low, high and volume from the Russell 2000 Index, while also including the metadata categories of market cap and sector. The data is daily ticker data for all the members of the index, from 2010 until the end of 2021.

Format of the data in excel is seen in figure 1. Note that running the code in excel is not great, but it shows the formatting previous to adjustments done in the project.



	A	B	C	D	E	F	G	H	I	J	K
1		AAN UN Equity					AAON UW Equity				
2	Dates	PX_LAST	PX_OPEN	PX_HIGH	PX_LOW	PX_VOLUMI	PX_LAST	PX_OPEN	PX_HIGH	PX_LOW	PX_VOL
3	12/23/2021	24.49	24.83	24.83	24.49	46526	78.57	78.9	79.67	78.46	3911
4	12/22/2021	24.53	24.2	24.54	23.91	55663	78.6	78	78.89	77.9	510
5	12/21/2021	24.22	23.84	24.47	23.84	79388	77.91	77.54	78.37	77.01	572
6	12/20/2021	23.45	23.65	23.65	22.96	106915	77.02	77.3	77.61	75.42	9861
7	12/17/2021	24.05	23.55	24.15	23.37	694523	77.89	77.26	79.16	76.03	4986
8	12/16/2021	23.55	23.94	24.08	23.48	138648	77.41	79.13	79.72	77.16	702
9	12/15/2021	23.72	23.32	23.76	22.69	148020	78.81	78.93	79.2	77.4	9101
10	12/14/2021	23.56	23.3	24.12	23.3	190754	78.81	80.49	81.1	78.62	694

Figure 1: Dataset as seen in excel. Shows the formatting previous to ingestion and editing in python.

This data is loaded into python pandas data frame, then reframed into a multiindex of the dates and the tickers.

Outlier Control

Remove stocks with insufficient observations (however many years required) in the index. This is to avoid the risk of outside influences causing inconsistency (especially since the Russel is based on market-cap size). Then, limit the dataset to include only a certain number of stocks with the highest market cap (number subject to change).

IMPROVEMENTS TO OUTLIER AND DATASET CONTROL:

- only the training period to remove data to avoid lookahead bias
- do further research into the flagged data points, then decide to change, adjust, or remove them based on the legitimacy of the discrepancy
- increase the stock universe

Basic Factor Engineering

Using TA-LIB (a python module), using the ingested data we can create:

- RSI index
- BBands (high and low)
- SAR (technical indicator to determine the direction that an asset is moving)
- Compute average true range (market volatility indicator)
- Moving average convergence and divergence (MACD) for momentum indicator
- Compute the historical returns and its deciles. Also create sector return deciles. (chose the lengths that you want for this, in this example using 1-, 5-, 10-, 21-, 42-, and 63-day returns).
- Compute the Normalized Average True Range (normalized ATR since it is better for comparison between different stocks)
- Compute forward returns (1-, 5- and 21-days). Not a feature, but important for testing and also computing factored returns for assessing the model

Also create time categorical variables for year, month, and weekday for the model to train off as well. Note that the implementation in LightGBM (used for this ML model) groups the levels of the categorical features to maximize homogeneity (or minimize variance) within groups with respect to the outcome values.

Trading Signals, Cross Validation

Using data created, select the training and validation periods. Create a dataset of the forward returns and create a separate set for the features. The model will want to predict 1, 5 or 21-day returns. We will cross-validate the model for different hyperparameter combinations, testing variables (like lookahead and lookback) then logging their performance so the best performing combinations can be chosen later. The performance of the models is measured by using an information coefficient based on spearman rank-order correlation.

The model will be using 4.5 and one year as the length of the training period. The test period is three-months long. (subject to adjustment) Since the model uses two years for validation, the three-month test period will have 8 folds (for the time series cross-validation).

Light Gradient Boosting Machine Model Tuning

Select the mode constraints on structure of each of the regression trees. So, set options for max_depth, number of leaves options (which is 2^(max_depth)), and the minimum data included in the leaves. Can also set different learning rate options, and also feature fraction options for the random feature selections.

Note that the loss function within the model also uses a spearman rank-order correlation to measure prediction accuracy.

The cross-validation iterates over six cross-validation configurations and collects the metrics. The information coefficient is recorded per day and computed during the cross-validation. The CV class permits us to:

- Train the model on a consecutive sample containing train_length days for each ticker.
- Validate its performance during a subsequent period containing test_length days and lookahead number of days, apart from the training period, to avoid data leakage.
- Repeat for a given number of n_splits while rolling the train and validation periods forward for test_length number of days each time.

Hyperparameter Impact from Cross Validation Results

To understand if there's a systematic, statistical relationship between the hyperparameters and the outcomes across daily predictions. So, we run a linear regression using various LightGBM hyperparameter settings as dummy variables and have the daily IC as the outcome to receive the results in figure 2.

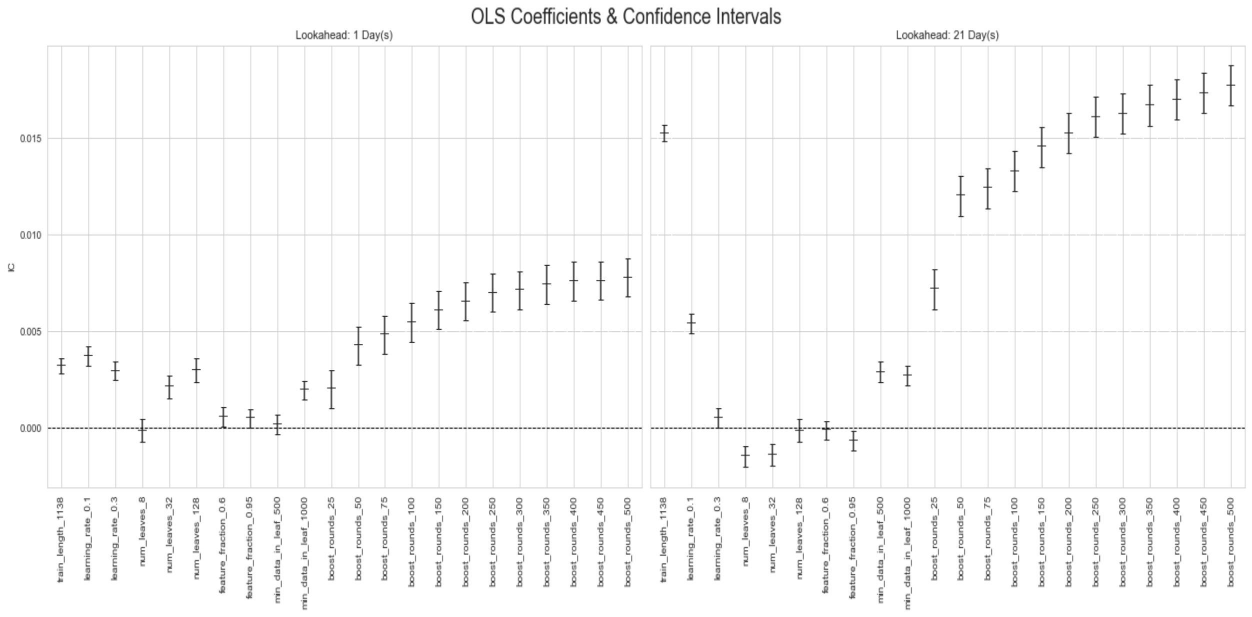


Figure 2: Linear regression measuring the hyperparameter influence on information coefficient

The chart shows the coefficient estimates and their confidence intervals for the two different forecast horizons. From here, you can see the results and reasoning for which hyperparameters result in the best outcomes.

From here, we find the top-preforming models and which hyperparameters for the three different prediction horizons (ranked by information coefficient).

We also retrieve the predictions for the top 10 validation runs and the selected lookahead period. This is because boosting methods train base estimators sequentially with the specific goal of reducing the bias of the combined estimator. The motivation is to combine several weak models into a powerful ensemble.

Computing Feature Importance

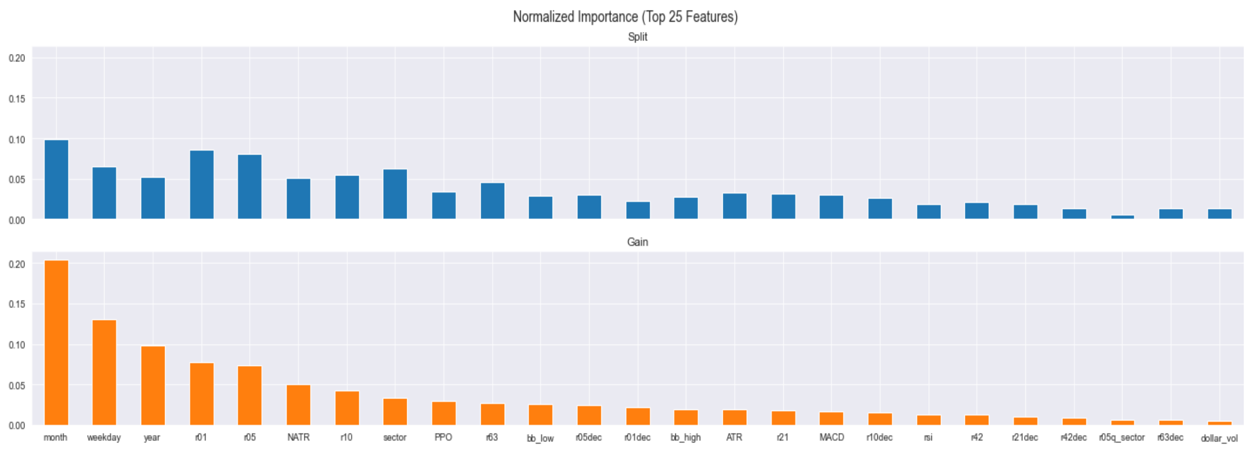


Figure 3: Basic feature importance. Does not account for interaction between different elements, a large blind spot for understanding a regression tree

Figure 3 shows the importance of the top 25 features. Values are created using feature_importance() method and the corresponding importance_type parameter.

For explaining the contribution of individual features to the output of tree ensemble models, the use of SHAP values is applied (more on SHAP later). SHAP provides differentiated insights into how the impact

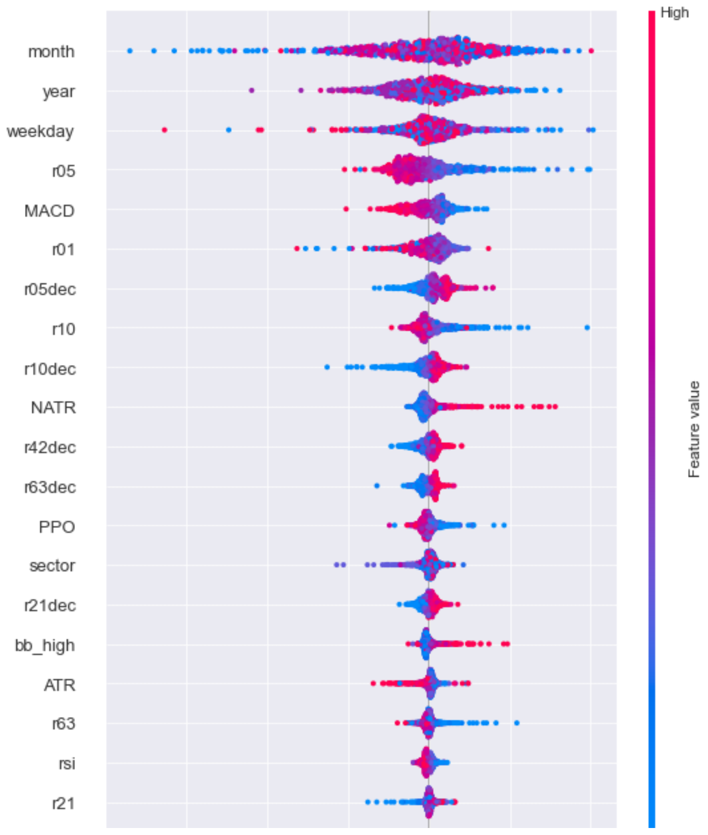


Figure 5: SHAP breakdowns for how values are interacting with the model impact. Red dot means positive impact, and blue dot means negative.

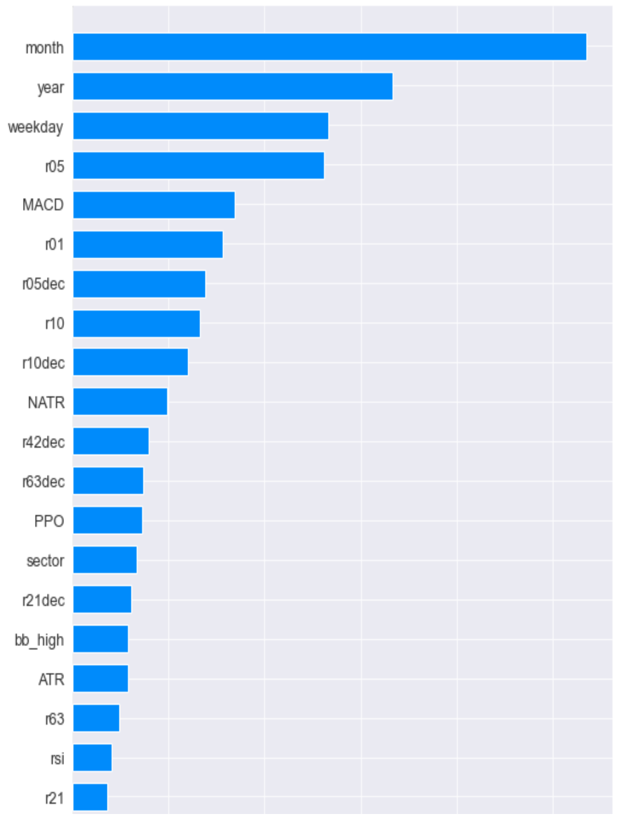


Figure 4: SHAP adjusted feature importance. Note the difference from the basic feature importance calculation.

of a feature varies across samples. Good for debugging the models and explaining why the model is making certain decisions. (They are better than partial dependence plots because they consider the and's and or's that your model may be doing with all these interconnected variables).

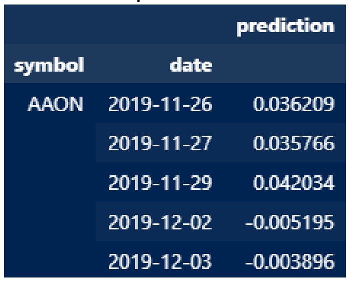
Making Out of Sample Predictions

The process for making the out of sample predictions will be similar to creating the single learner in the previous section. Using the cross-validation results, we select the top models' hyperparameter combinations for the chosen lookahead period you want to test. This example will be creating predictions for two years outside of the test set.

The model will iterate over the data set, splitting the data into test sets of the same length as the hyperparameters dictate. For example, the best performing model from the CV results has a 2-month test period, so it will iterate and train 8 times and generate predictions across the next period. The model is basically forecasting on a rolling origin.

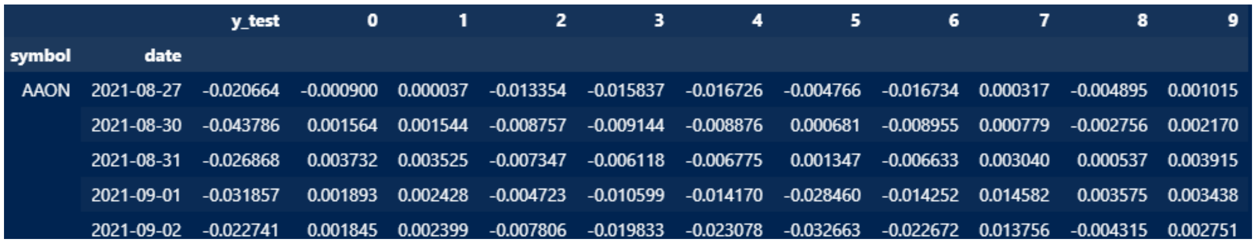
Then, this process is repeated for however many models you want to implement and create signals that you can trade upon. You can also use different lookahead settings, but for simplicity the model used in this project only bases it off a single lookahead setting.

Also, for simplicity, the signal will be a simple average of the models' predictions. The output for this will look like figure 7, and the data frame of the different models separated is in figure 6. The next data frame, seen in figure x, is that of the combined predictions, which will eventually be used to trade upon.



		prediction
symbol	date	
AAON	2019-11-26	0.036209
	2019-11-27	0.035766
	2019-11-29	0.042034
	2019-12-02	-0.005195
	2019-12-03	-0.003896

Figure 7: Dataset snippet from the combined predictions, or the ML 'signal'

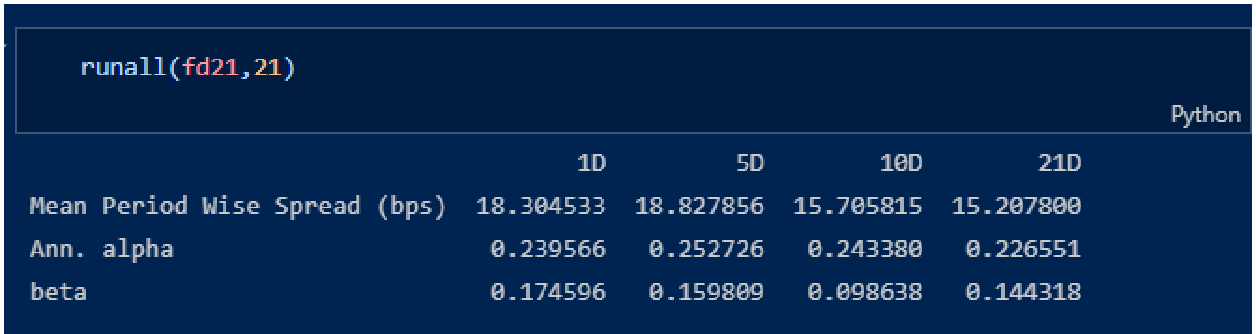


		y_test	0	1	2	3	4	5	6	7	8	9
symbol	date											
AAON	2021-08-27	-0.020664	-0.000900	0.000037	-0.013354	-0.015837	-0.016726	-0.004766	-0.016734	0.000317	-0.004895	0.001015
	2021-08-30	-0.043786	0.001564	0.001544	-0.008757	-0.009144	-0.008876	0.000681	-0.008955	0.000779	-0.002756	0.002170
	2021-08-31	-0.026868	0.003732	0.003525	-0.007347	-0.006118	-0.006775	0.001347	-0.006633	0.003040	0.000537	0.003915
	2021-09-01	-0.031857	0.001893	0.002428	-0.004723	-0.010599	-0.014170	-0.028460	-0.014252	0.014582	0.003575	0.003438
	2021-09-02	-0.022741	0.001845	0.002399	-0.007806	-0.019833	-0.023078	-0.032663	-0.022672	0.013756	-0.004315	0.002751

Figure 6: The data frame snippet showing the ten different model predictions. (where header x+1 is the model ranking from the cross-validation results).

Using Alphalens to Assess the Signal as an Alpha Factor

Using the Alphalens module, we can break down how the factor effective the factor signal is (in a closed environment, not equivalent to a live backtest). We create a signal factor, and the module sorts the signals into five different quantiles based on the signal strength.



```
runall(fd21,21)
```

Python

```
                                  1D         5D        10D        21D
Mean Period Wise Spread (bps)  18.304533  18.827856  15.705815  15.207800
Ann. alpha                      0.239566   0.252726   0.243380   0.226551
beta                            0.174596   0.159809   0.098638   0.144318
```

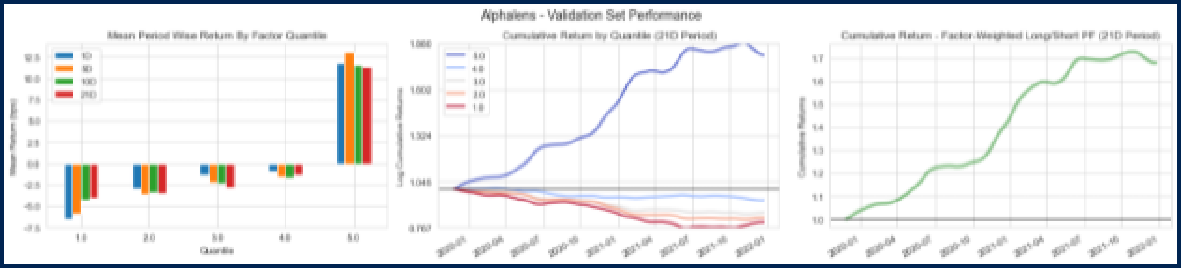


Figure 8: Alpahlens factor returns breakdown. Shows results for trading on different horizons, as well as showing some cumulative returns for simple trading on the model

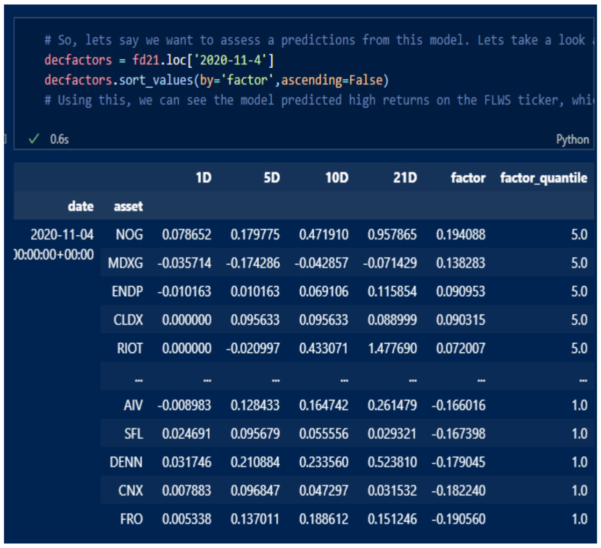
Here we can see the breakdowns of the model's different quantiles, where the 5th quantile has the highest signal, 1st the lowest, and ext. Alphalens also computes a factor weighted long/short strategy to show the returns of the factors over the period. Overall, you can see that the model has a decent showing, with a consistent quantile breakdown (increasing returns as you move into higher quantiles). A factor-weighted long/short implementation provides an annual alpha is 0.22 and a beta of .144 (for the 21D period signal). So, we can see that there is some validity to how the signal is predicting returns. But it is very important to acknowledge the alpha and beta here do not point to actual portfolio performance for many reasons, so the backtest will provide a better insight into this.

Understanding the Model

Now that the model seems to be doing well, we want to dig into some of the patterns that the model is using for predictions. We do this by digging into single predictions. Here we have the forward factor returns from the predictions set for November 4th, 2020. The highest factor the model has is for 'NOG' (Northern Oil and Gas). We can see the returns for 1D, 5D, 10D, and 21D. The factor of .19 is the average of the top ten models' predictions. We can also grab one of the LGB models and break down this example to see what is moving the asset in the positive direction. To do this with a complex regression tree system SHAP values are a great resource. The values are an adaptation of game theory concepts on a tree-based model, which have been shown to be consistent and theoretically optimal (as well as being computationally efficient). They provide a quick implementation and breakdown of previously bulky complex decisions trees like this one.

So, we take the predictions for 'NOG' from the top model and the current iteration fold it was in and compute the SHAP values for the prediction.

The biggest SHAP feature value that is moving this certain prediction is the fact that the NATR (Normalized Average True Range) indicator. So, the model is sensing that the volatility for the previous day was very high, which moves its prediction in a positive direction. Also, it sees that the momentum (as measured by the MACD) and the 63-day return is negative, which push the factor to predict a more positive result for its future returns. The SHAP values are not a perfect indicator of what the model is thinking, but at least provides insight into why a certain prediction would be so high. This is also important to look at if you are assessing the model for over-emphasizing noise factors, which accounting NATR into a positive return prediction can point to.



```
# So, lets say we want to assess a predictions from this model. Lets take a look
decfactors = fd21.loc['2020-11-4']
decfactors.sort_values(by='factor',ascending=False)
# Using this, we can see the model predicted high returns on the FLWS ticker, whi
```

✓ 0.6s Python

date	asset	1D	5D	10D	21D	factor	factor_quantile
2020-11-04 00:00:00+00:00	NOG	0.078652	0.179775	0.471910	0.957865	0.194088	5.0
	MDXG	-0.035714	-0.174286	-0.042857	-0.071429	0.138283	5.0
	ENDP	-0.010163	0.010163	0.069106	0.115854	0.090953	5.0
	CLDX	0.000000	0.095633	0.095633	0.088999	0.090315	5.0
	RIOT	0.000000	-0.020997	0.433071	1.477690	0.072007	5.0
	...	...	...	...	...	...	...
	AIV	-0.008983	0.128433	0.164742	0.261479	-0.166016	1.0
	SFL	0.024691	0.095679	0.055556	0.029321	-0.167398	1.0
	DENN	0.031746	0.210884	0.233560	0.523810	-0.179045	1.0
	CNX	0.007883	0.096847	0.047297	0.031532	-0.182240	1.0
	FRO	0.005338	0.137011	0.188612	0.151246	-0.190560	1.0

Figure 9: Dataframe of the factor breakdowns from Alphalens. Shows the returns for the stock for the different lookaheads

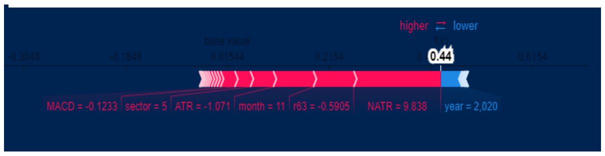
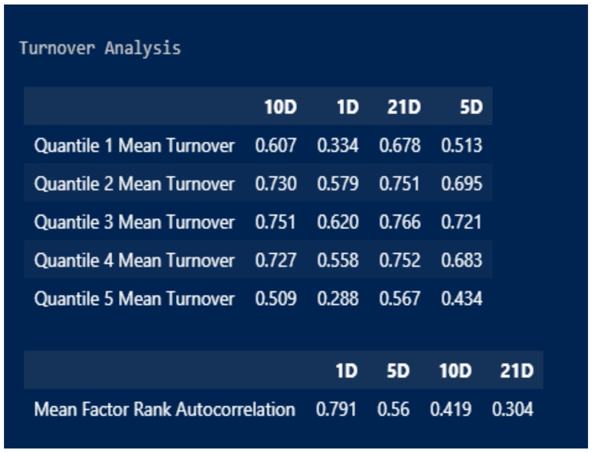


Figure 10: SHAP value force plot for 'NOG' on November 4th, 2020



Turnover Analysis

	10D	1D	21D	5D
Quantile 1 Mean Turnover	0.607	0.334	0.678	0.513
Quantile 2 Mean Turnover	0.730	0.579	0.751	0.695
Quantile 3 Mean Turnover	0.751	0.620	0.766	0.721
Quantile 4 Mean Turnover	0.727	0.558	0.752	0.683
Quantile 5 Mean Turnover	0.509	0.288	0.567	0.434

	1D	5D	10D	21D
Mean Factor Rank Autocorrelation	0.791	0.56	0.419	0.304

Figure 11: Turnover analysis from Alphalens module

Some of the bigger problems with the current state of the algorithm is that the quantile turnover (how much assets move around the quantiles). This can point to having high transaction costs if you are constantly readjusting to fit your factor's predictions. Also, the model seems to leverage things from the volatility indicators to gain more profit, which can expose the signal to great amounts of risk. You would want to analyze trading the signal in large downturns to avoid large drawdowns. Finally, the fixed-point nature of running these models can sometimes cause the model to again fit to noise, which would cause problems trading within a live environment.

Creating and Implementing a Backtest with the Signal

Implementing a backtest is the next step since it is important to see how a model performs in a more live environment while also incorporating commission fees into account. The backtest uses the same format of open, high, low, and close data. The predictions which were produced earlier are used to create a simple long-short strategy, where the highest signals on a given day are long traded, and the lowest ones shorted. It will run for a year, and in turn return portfolio metrics for the period using the strategy. Each trade the model makes is saved, which allows for deeper inspection into the model.

Backtest Results

The model had a large amount of turnover when running the backtest, which is something that happens with such a simple model. Thus, even increasing the commission fee slightly would change the backtest results by a great amount. Here we can see the difference in returns when applying a 1.5% (figure 12) and 1% fixed commission (figure 13) on the backtest results.

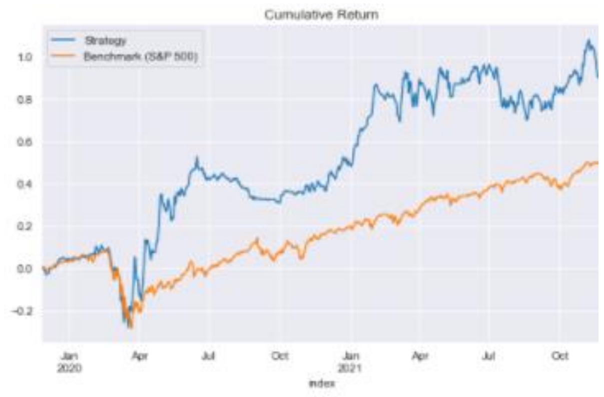


Figure 12: 1.5% commission fee backtest results

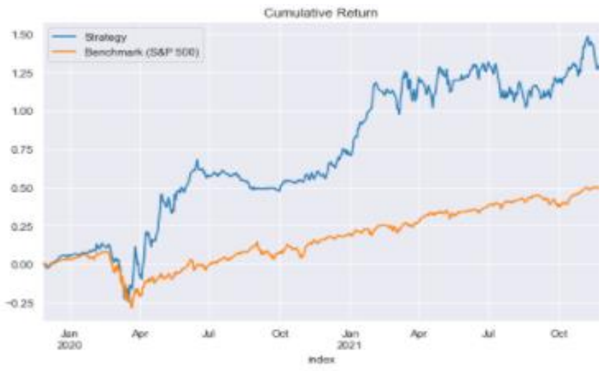
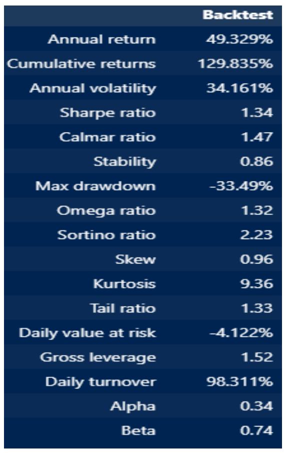


Figure 13: 1% commission fee backtest results

Also, the model did not outperform the benchmark during the Covid-19 drawdown period, and if anything, probably did slightly worse which is not a good sign for the overall validity of trading with the signal. This is a general trend, there tends to be a large exposure to volatility and drawdowns (which is seen with the high annual volatility and low stability in the portfolio breakdowns). There is also no risk adjustment for the backtest, so it is not surprising that drawdowns would be so large. Despite this, annual returns of almost 50% and a Sharpe ratio of 1.34 with such a simple strategy is a good sign. In a model with such high volatility, the Sortino ratio is a good indicator to look at as well, since it is an adjusted version of the Sharpe ratio which only penalizes downward deviations when accounting for volatility. A value of 2.23 shows that the drawdowns are not the main driver of the high volatility metrics in the results. This fact points again to a decent signal for trading.



	Backtest
Annual return	49.329%
Cumulative returns	129.835%
Annual volatility	34.161%
Sharpe ratio	1.34
Calmar ratio	1.47
Stability	0.86
Max drawdown	-33.49%
Omega ratio	1.32
Sortino ratio	2.23
Skew	0.96
Kurtosis	9.36
Tail ratio	1.33
Daily value at risk	-4.122%
Gross leverage	1.52
Daily turnover	98.311%
Alpha	0.34
Beta	0.74

Figure 14: portfolio statistics from the backtest

The next steps for testing the validity this model would be to adapt the model to create predictions in a paper traded environment while also implementing a more adjusted trading strategy to reduce the turnover and exposure to risk.

Conclusions

The overall testing of the factor shows decent results, even with simple implementation. The model creates some sort of alpha which could be traded on. Despite this, running a model fully reliant on this factor traded model is likely not a great strategy, especially since ML models tend to fall very flat against the effects of arbitrage, which testing in a frozen state like this does not account for. Also, it would be incredibly important to stress test a model like this, since they are not good at adjusting to changing markets or downturns. This effect was already seen with how it did not predict returns during the COVID drawdown particularly well. Leveraging machine learning would be better for adjusting to things like risk management or using it along with a more proven and 'solid' strategy.