Capstone 3 Final Report:

Constructing a futures trading strategy which combines custom asymmetric objective functions and direct position optimization

Matt Kosik

Problem Statement:

The implementation of any trading strategy can be broken down into two separate processes: forecasting and position sizing. Forecasting involves predicting the future prices of a particular asset or combination of assets based on the certain information set known at the present. The position sizing function takes the forecasts as an input, combines them with investment goals (such as risk and return targets and available capital), and then outputs a buy or sell signal with an associated magnitude that represents the desired position size in the assets to be traded.

I am proposing a new procedure that improves upon both steps of this process. First, I will introduce and implement a custom asymmetric objective function in the forecasting step which incorporates real world consequences of errors into predictions. Secondly, I will incorporate a method that combines the forecasting and positing sizing steps into a single direct position optimization. This is akin to training forecasting and posting sizing models simultaneously rather than in a step wise manner. The overall trading strategy has highly attractive out of sample results across a variety of measures and looks promising for further research.

Data:

The data used in this project are all historical asset returns. I chose to create this trading strategy in the futures market because it has a significant amount of historical data, is heavily studied in academic literature, and is extremely liquid. All data can be found in the *pysystemtrade* python package.

I use daily returns data from the period 1-Jan-2000 to 1-Jan-2020 across the following futures markets:

FX:

AUD, EUR, GBP, JPY, MXN

Commodities:

Corn, Soybean, Wheat, Crude Oil, Natural Gas, Lean Hog, Live Cow, Copper, Palladium, Platinum, Gold

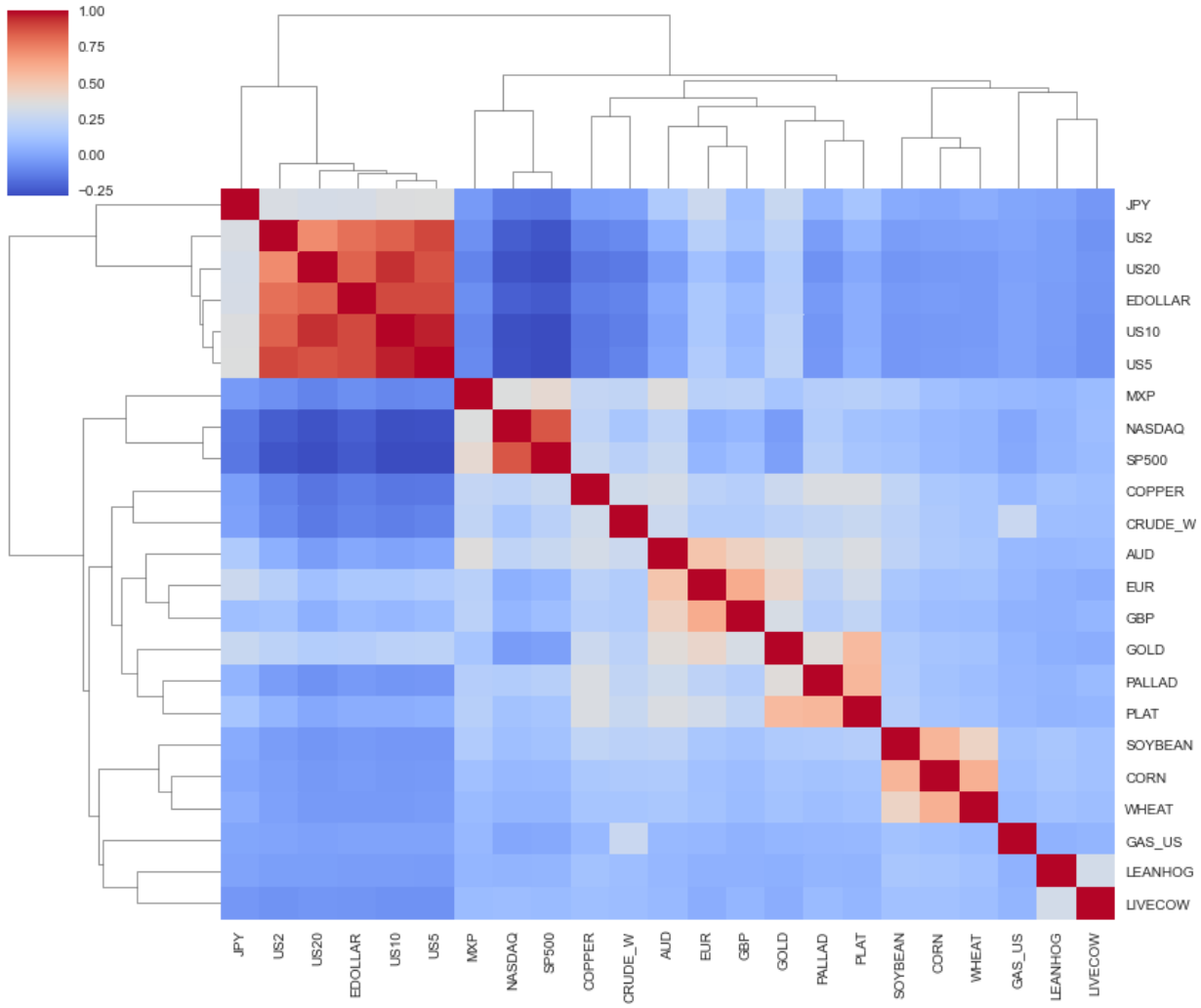
Equities:

Nasdaq, S&P500

Rates:

Eurodollar, US2y, US5y, US10y, US20y

As shown in the dendrogram correlation map below, there are small pockets of high correlation (within rate and equity products or similar commodities) but the assets chosen should generally provide strong diversification benefits.



As common in futures trading strategy literature, I will be scaling all returns to an ex-ante target annualized volatility of 15% prior to trading.

Future	Ann. Mean	Ann. Std.	Ann. Sharpe	Future	Ann. Mean	Ann. Std.	Ann. Sharpe
AUD	0.056216	0.144384	0.389349	**MXP**	0.057817	0.143863	0.401891
COPPER	0.0389	0.14395	0.270233	**NASDAQ**	0.103229	0.143178	0.720986
CORN	-0.017191	0.143293	-0.119973	**PALLAD**	0.079726	0.14506	0.549608
CRUDE_W	0.091488	0.144402	0.633565	**PLAT**	0.0721	0.144754	0.498087
EDOLLAR	0.10207	0.144187	0.707903	**SOYBEAN**	0.071745	0.143631	0.499507
EUR	0.008412	0.144293	0.0583	**SP500**	0.088901	0.142701	0.622985
GAS_US	-0.040018	0.144611	-0.276729	**US10**	0.112837	0.14426	0.78218
GBP	0.013024	0.1457	0.089392	**US2**	0.090295	0.142302	0.63453
GOLD	0.093991	0.142983	0.657355	**US20**	0.105059	0.144551	0.726799
JPY	-0.071964	0.175916	-0.409083	**US5**	0.106349	0.14388	0.739152
LEANHOG	0.052851	0.144687	0.36528	**WHEAT**	-0.011832	0.144365	-0.08196
LIVECOW	0.017509	0.144538	0.121138				

Features:

To keep the model as simple as possible, I will only be using traditional trend and momentum features based on the historical price series.

Vol normalized MACD(x,y) for x,y time-spans = (8,24), (16,48), and (32,96)

Vol normalized trend for lookbacks of: 1 day, 1 month, 3 months, and 6 months

*Regression adjusted momentum: R^2 * beta, both from exponential regression of rolling past prices*

An example for gold:

DATETIME	GOLD_macd1	GOLD_macd2	GOLD_macd3	GOLD_trend1	GOLD_trend2	GOLD_trend3	GOLD_trend4	GOLD_mom
12/25/2019	-0.06304	0.17909	2.48661	1.40385	1.18672	-0.13566	0.63625	0.00017
12/26/2019	0.10025	0.27515	2.60833	0.48150	1.25777	0.00000	0.83141	0.00061
12/27/2019	0.30332	0.38970	2.63404	1.35351	1.69217	0.73362	0.87295	0.00150
12/30/2019	0.48787	0.51105	2.75770	-0.04374	1.54933	0.46654	1.25723	0.00138
12/31/2019	0.69820	0.64324	2.79122	1.42152	1.90869	0.27778	1.39691	0.00155

When predicting the direct positions for the entire portfolio at once, the feature size will expand from the above to 8*total number of assets in the portfolio. All features and normalized prior to training.

Timeframe:

I will be constructing the models to follow an expanding window training set and fixed test set. This strategy retrains and revalidates the models every 4 years to incorporate all available information up to that point in time. It will then predict a trading strategy for the following 4 years.

Model:

The first part of my proposed strategy is improving the forecasting step by incorporating custom loss functions. This is an important departure from most financial forecasting strategies which often use a symmetric Mean Squared Error (MSE) objective function. MSE has historically been used due to its neat mathematical properties which drastically simply the operational aspects of the calculation and maximize the likelihood of the model under certain assumptions. This is a perfectly fine solution for forecasting on its own but breaks down when you consider the decisions that are based on the forecasts.

This is best shown through a simple example. Assume your forecasting model predicts that a certain asset will return +1% over the investment period.

Now consider two scenarios:

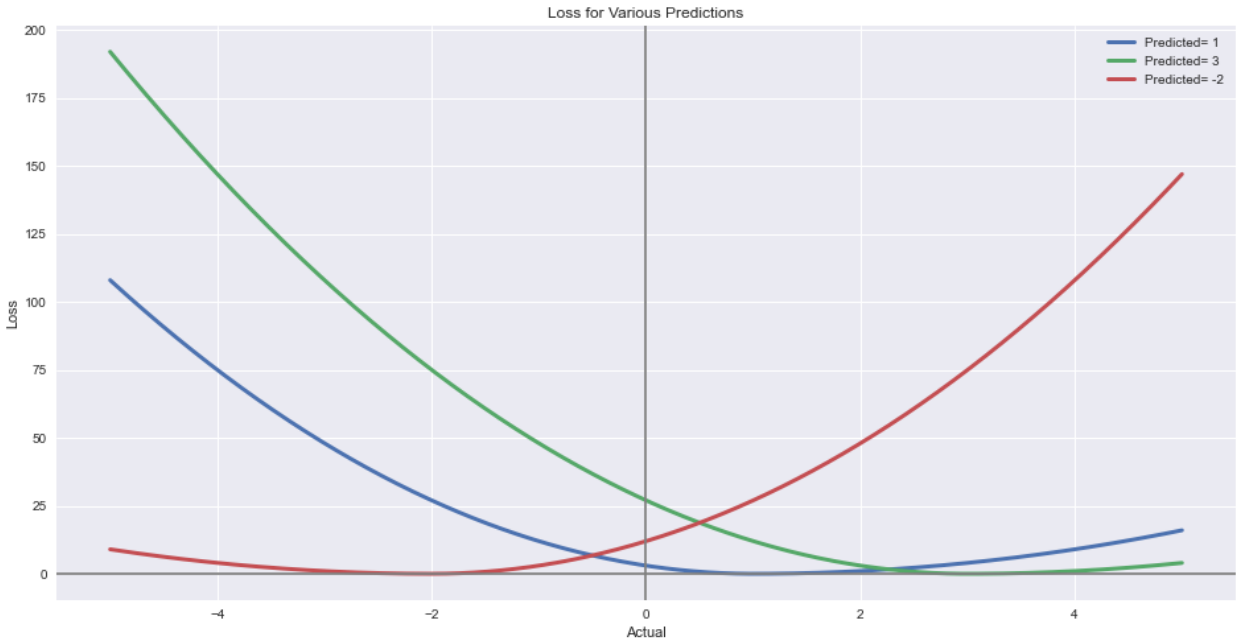
1. The actual return is +3%
2. The actual return is -1%

In both scenarios, the squared error is $(3\% - 1\%)^2 = (-1\%-1\%)^2 = +.04\%$. This symmetry is preferred from a forecasting standpoint but as investors, we obviously have a higher preference for the +3% return over the -1% return. Therefore, it is likely better to use an asymmetric objective function. For this project, I've simply used the below formula to keep the function smooth.

$$L(\hat{y}) = \begin{cases} (\hat{y} - y)^2 & if\ (\hat{y} * y) > 0\ and\ |y| > |\hat{y}| \\ c * (\hat{y} - y)^2 & if\ (\hat{y} * y) \leq 0\ and\ |y| < |\hat{y}| \end{cases}$$

Where $\hat{y}$ = estimated value, y = actual value, and c = some constant

This results in the desired asymmetric behavior. As you can see with the blue line, if we predicted +1 as in our example above, the loss would be greater for -1 than +3.



While this result is much preferred to the traditional MSE approach, it significantly increases the operational workload. Fortunately, some of the desired models have python packages which allow for custom objective function definition.

The second part of my proposed procedure is improving the position sizing step. Again, I will start by addressing what has traditionally been done and the current disadvantages. Once a forecast is complete, there needs to be some translation into the traded position. The most simple and common way this is currently done is to take the sign of the forecast and translate that into a long or short position. This +/- 1 is then scaled to match the desired volatility and reflect the investor's bankroll. The main problem with determining the position size this way is that it discards precious forecasting information about the magnitude of the forecasted return. Given a bound of [-1,1] on position size, the ideal model would output an optimal position somewhere within that interval and not be limited to the extreme binary cases.

In trying to create such a model using supervised learning techniques, we quickly run into the problem that our returns data aren't labeled for best ex-ante position size and we'd be forced to use the

extremes. We can get around this issue by having the model directly output a position size that is optimized for a maximum performance metric. I take this a step further by optimizing for an asymmetric performance metric, specifically the Sortino Ratio (mean return / downside deviation of returns) which should provide even more desirable results.

Taking the general strategies mentioned above into consideration, I decided to implement them across an ensemble of carefully chosen models. The models were selected to combine a variety of linear and non-linear approaches as well as the capability to include custom loss functions.

In total, I have used 6 models. The final ensemble model will be an equal weight combination.

1. Forecast: Ordinary Least Squares regression, mean squared error; Positioning: Binary Sign
2. Forecast: LightGBM regression, custom asymmetric error; Positioning: Binary Sign
3. Forecast: LSTM regression via Keras, custom asymmetric error; Positioning: Binary Sign
4. Forecast: All Assets Random Forest regression, mean squared error; Positioning: Binary Sign
5. Direct Positions: Linear regression via Keras, custom Sortino error
6. Direct Positions: LSTM regression via Keras, custom Sortino error

Model Specifications:

Model 1: no hyperparameters

Model 2: Random Grid Search CV, 25 iterations over 5 rolling windows using sklearn TimeSeriesSplit

Params = {"num_leaves" : [15,30,50,100], "max_depth" : [-1,2,6,10,15,25], "learning_rate" : [0.1,.001,.0001], "n_estimators" : [25,50,100,150]}

Model 3: LSTM window = 50, single layer LSTM cells = 20, dropout = .3, recurrent_dropout=.3, validation split = .25, optimizer = Adam, epochs = 100, batch_size = len(data)/5, calback to best val_loss

Model 4: Random Grid Search CV, 25 iterations over 5 rolling windows using sklearn TimeSeriesSplit

Params = { "n_estimators" : [25,50,100,150], "max_depth": [2,6,10,15,25], "min_samples_split": [2, 5, 10, 15, 100], ”min_samples_leaf" : [1, 2, 5, 10]}

Model 5: validation split = .25, optimizer = Adam, epochs = 500, batch_size = len(data)/5, calback to best val_loss

Model 6: LSTM window = 50, single layer LSTM cells = 20, dropout = .3, recurrent_dropout=.3, validation split = .25, optimizer = Adam, epochs = 100, batch_size = len(data)/5, calback to best val_loss

Note: For LSTM models, training size was reduced by window length in order to have same number of predictions as other models.

Results:

After training all 6 models and testing them on the out of sample data, I began to see some interesting results.

The benchmarks represent very simple trading strategies:

Long = Always long

MACD = For each individual asset, max long when average macd signal >0, max short otherwise

Trend = For each individual asset, max long when average trend signal >0, max short otherwise

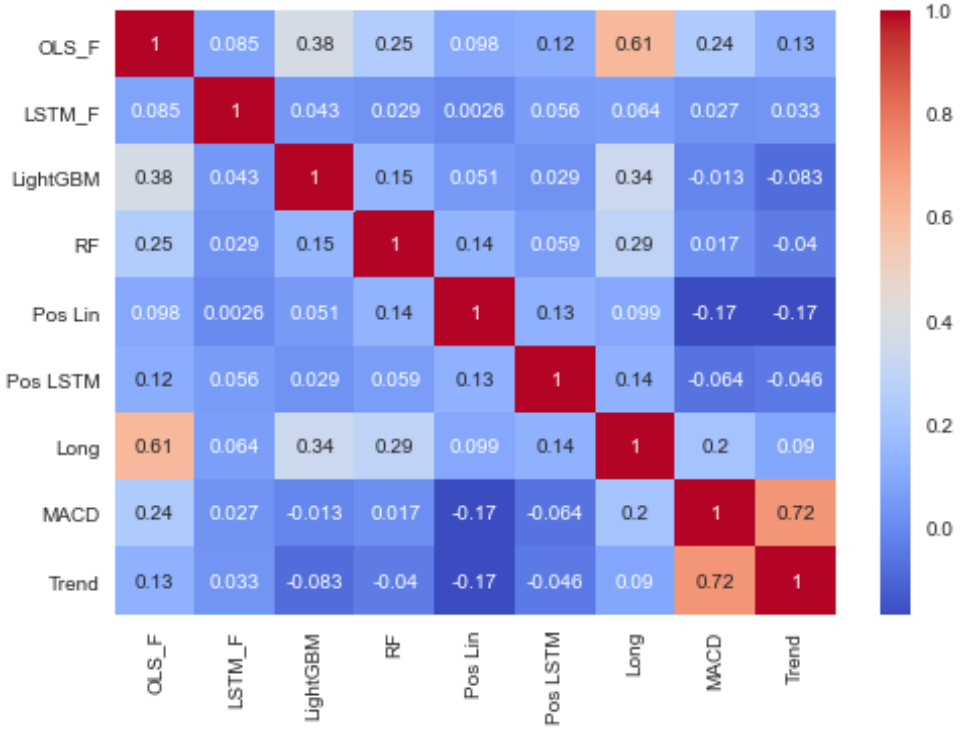
Strategies (Annualized Results):

	OLS_F	LightGBM	LSTM_F	RF	Pos Lin	Pos LSTM	Combined
Avg. Returns	3.71%	2.15%	1.60%	3.94%	1.30%	0.93%	**2.27%**
Std.	4.06%	3.38%	2.97%	4.77%	2.34%	1.04%	**1.69%**
Sharpe	0.915	0.637	0.538	0.826	0.553	0.902	**1.343**
Sortino	1.339	0.927	0.773	1.227	0.818	1.314	**2.063**
Max Drawdown	-8.86%	-8.47%	-6.92%	-17.10%	-8.67%	-3.02%	**-3.32%**

Benchmarks (Annualized Results):

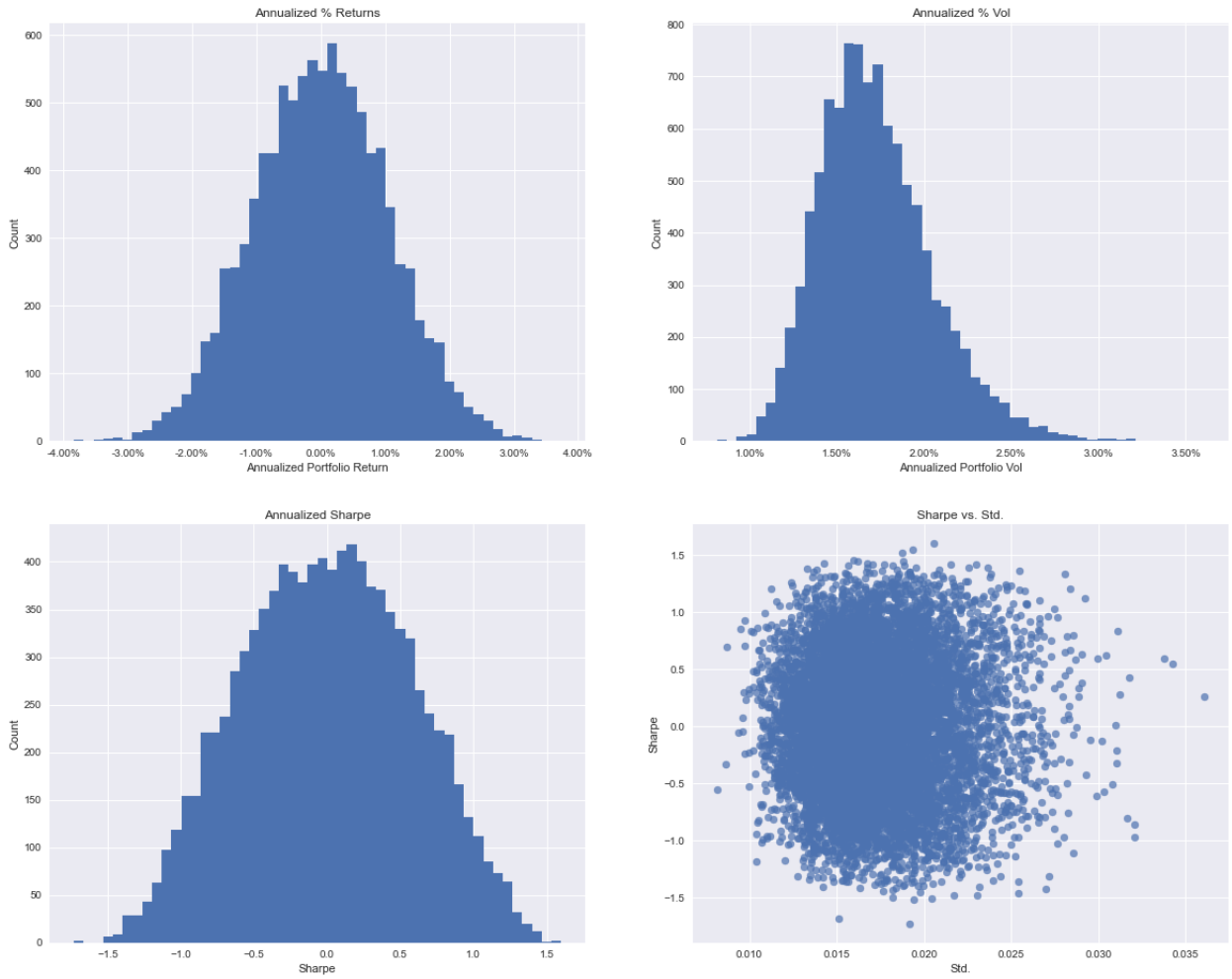
	Long	MACD	Trend
Avg. Returns	4.57%	5.21%	6.04%
Std.	6.06%	5.03%	4.97%
Sharpe	0.754	1.035	1.216
Sortino	1.102	1.530	1.843
Max Drawdown	-19.78%	-8.54%	-5.98%

Correlation:



All 6 individual models have positive returns with attractive alternative metrics. But what I found most surprising was the low correlation between models, especially when considering all models were trained on the exact same features. This highlights the benefit of the final combined ensemble model. The combined approach has the best Sharpe and Sortino across the board and nearly the best drawdown. Another clear trend is that the 2 direct position optimization strategies significantly reduce the standard deviation of the returns. Furthermore, this method produced some of the lowest and sometimes even negative correlations to the rest of the strategies.

To get an alternative view on the success of my strategy, I ran a Monte Carlo analysis generating uniform [-1,1] weights for all assets in the portfolio. I ran this simulation 10,000 times across 4,000 actual days randomly chosen with replacement from the realized out of sample returns.



The returns and volatilities of my strategies look inline with the simulated results. Importantly, the Sharpe of my combined strategy exceeds the 99% threshold of simulated results of 1.23 showing statistical significance of outperformance at that level.

Conclusion:

The proposed ensemble trading strategy was highly successful, outperforming the standard benchmarks and producing very good portfolio statistics. I showed how asymmetric loss functions and direct

position optimization could be incorporated into investment forecasting and construction of a trading portfolio. These methods created strategies with positive returns and very low correlation which provides fantastic diversification benefits.

Further Research and Application:

This project was a great first step into building a successful trading strategy. While the approach was simplified to a large extent, the general theories can be easily expanded upon and adapted to live trading. One of the biggest assumptions ignored here was trading costs. A more complex strategy likely has higher turnover so this will need to be investigated over time. Also, further feature engineering would certainly be useful, as I used very simple and likely overmined features.

I think the biggest area for further research is the direct position optimization. As observed over the course of the project, this step essentially combines forecasting with portfolio optimization. Optically, the method seems to be focused on minimizing the realized standard deviation of returns. It would be interesting to build a natural floor for standard deviation into the objective function in order to compare if the results still hold for higher volatilities.

Interestingly, half of the models were negatively correlated with the 2 most commonly used momentum trading strategies today, macd and trend. In fact, if I include these 2 benchmarks into my combined model, the Sharpe jumps up to an excellent 1.80. This is an extremely good result and signals that existing strategies could benefit from being combined with my approach. An extension here would be to then create a stacked model which would optimize the weights within the ensemble instead of naively using an equal weighted approach.