Stock Price Prediction

May 2020

Data and Feature Selection

I have chosen to examine copper and platinum commodity futures contracts with respective tickers HG1 and PL1. These are generic front month instruments created by Bloomberg. I have chosen to use price adjustments at the roll to ensure that price levels are accurate. Daily opening, closing, high and low prices were downloaded from Bloomberg over the period 03/01/2000 - 30/04/2020 in USD. For reference, the fields used were $PX\_OPEN, PX\_LAST, PX\_HIGH$ and PX_LOW respectively.

Initial feature choices were:

- Log return over k days - $r_{t,k} := \log(p_t/p_{t-k})$, $k = 1, 2, 3, 4, 5$
- Signed log return over 1 day - $sr_t := \text{sgn}(\log(p_t/p_{t-1}))$
- Price momentum over k days - $m_{t,k} := p_t - p_{t-k}$, $k = 1, 2, 3, 4, 5$
- Price SMA over k days - $sma_{t,k} := (1/k)\sum_{i=0}^{k-1} p_{t-i}$, $k = 5, 10, 15, 20, 25$
- Price EWMA over k days - $ewma_{t,k} := \lambda\, ewma_{t-1,k} + (1-\lambda)p_t$ where $\lambda = 0.9$ and $k = 5, 10, 15, 20, 25$
- Volatility over k days - $\sigma_{t,k} := \sqrt{\frac{1}{k-1}\sum_{i=0}^{k-1}(r_{t,1} - \widehat{r}_1)^2}$, where $\widehat{r}_1 = (1/k)\sum_{i=0}^{k-1} r_{t,1}$ is the sample mean return over the window, $k = 20, 40, 60, 80, 100$
- Intraday Price Momentum - $im_t := \text{sgn}(close_t - open_t)$
- High-Low Price Momentum -

$$hlm_t := \begin{cases} 1, & \text{if } |close_t - high_t| < |close_t - low_t| \\ -1, & \text{if } |close_t - high_t| > |close_t - low_t| \end{cases}$$

The first 100 days of prices are removed so that all features are available over the same period. Days on which there is no price movement are also removed as suspected holidays. The choices of k were somewhat arbitrary and will certainly lead to redundancy for some features. In particular, note that only 1 choice of

window length is required for the EWMA price average. This follows from the recursive definition - the difference between two EWMAs with different window lengths will decay geometrically in the 'persistence' term, λ. In symbols:

$$|ewma_{t+1,k} - ewma_{t+1,l}| = \lambda\,|ewma_{t,k} - ewma_{t,l}|$$

A value of λ very close to 1 could be chosen to slow the decay but this would lead to an almost constant series. In addition, SMAs of differing window lengths are likely to be quite highly correlated with one another, momentum and returns are closely connected and so on.

To begin to quantify feature similarity we can look at correlations. Figure 1 illustrates the correlation structure amongst the momentum features for copper. We see that as the date gap between prices widens, correlation between the simple price momentum measures decreases. The categorical features are also positively correlated with these simple momentum measures and with the same hierarchy, but at lower levels.

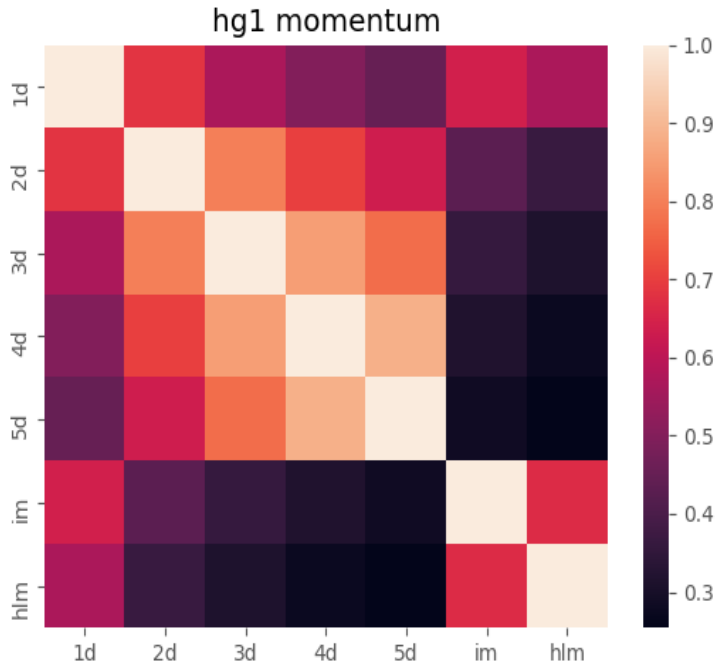

Figure 1: Copper Momentum Correlations

PCA

To get a more global picture of the possible redundancy of features we can look at a PCA of the feature space. I have computed the eigenvectors and eigenvalues of the correlation matrix of the features, which is equivalent to first standardising the data and using the usual covariance approach. This is necessary due to the large differences in variance amongst the various features. The contribution to variance from the jth eigenvector, v_j, can be measured by $\lambda_j / \sum_j \lambda_j$, where λ_j is the eigenvalue corresponding to v_j.

Figure 2 shows that for both copper and platinum almost 40% of the variance is explained by the first eigenvector and that the first 5 eigenvectors explain more than 90% of the overall variance. Furthermore, we see that by the 10th eigenvector we have almost entirely explained the variance of both commodities and so it seems reasonable to predict that only 5-10 features will be necessary for fitting the prediction model. One choice from each of the classes of features listed above may be an appropriate choice.

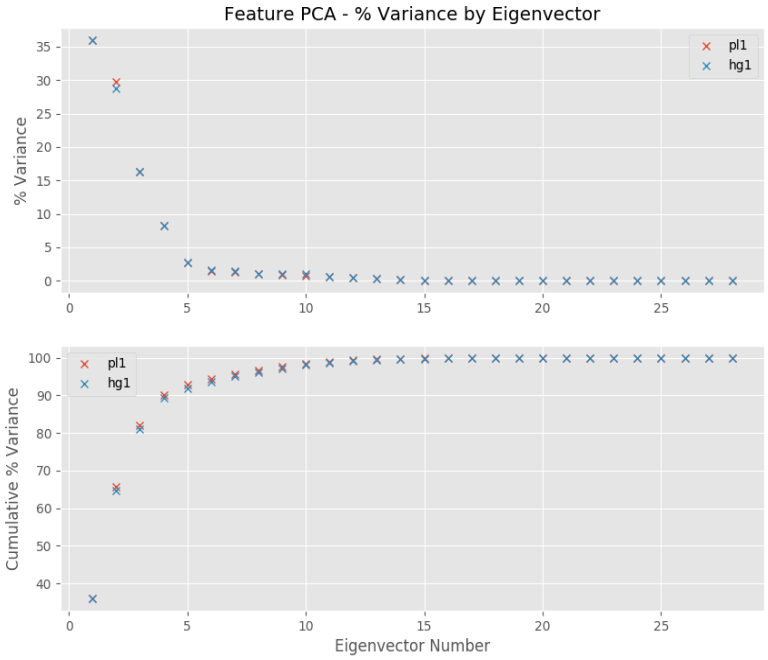

Figure 2: PCA - Contributions to Variance

Up vs Down Days

Table 1 displays the number and proportions of days with up and down moves for the two securities of interest. As expected, the directions are fairly evenly balanced and hence it is not necessary to apply any special stratified sampling techniques when training our models. Note that the benchmark accuracy to beat is higher for platinum than copper. This follows because we can expect 52.54% accuracy for a model that simply predicts platinum while go up every day, whereas the majority prediction strategy only ensures an accuracy of 50.50% for copper.

	Count	Up Moves	Down Moves	Up Ratio	Down Ratio
HG1	4958	2504	2454	50.50%	49.50%
PL1	4977	2615	2362	52.54%	47.46%

Table 1: Up and Down Moves

Forward Selection

To help with feature selection I implemented a basic forward selection algorithm. The below procedure, taken directly from [1], is followed with $p = 23$ features, having omitted the EWMA features as unhelpful.

- Let $\mathcal{M}_0$ denote the null model that contains no features
- For $k = 0, \ldots, p-1$
  - Consider all $p-k$ models that augment the features in $\mathcal{M}_k$ with one additional feature
  - Choose the best amongst the $p-k$ models and call it $\mathcal{M}_{k+1}$. Here best is chosen based on average accuracy from 5−fold cross validation
- Select a single best model from amongst $\mathcal{M}_0, \ldots, \mathcal{M}_p$ based on the accuracy measures mentioned above

Figure 3 shows the results of such a procedure run over the full data history on 75% of randomly partitioned test data for platinum. Classifiers were run with default parameters except logistic regression, for which I removed the penalty. Note that accuracy increases for all classifiers as more features are added, up until a point. Beyond this point there is a more rapid decay in performance. The optimal number of features discerned in this way varies by classifier type, with all but the k-Nearest Neighbours (kNN) classifier appearing to be most accurate with 5 or 6 features. In this case, kNN seems to improve until 13 features are being used.

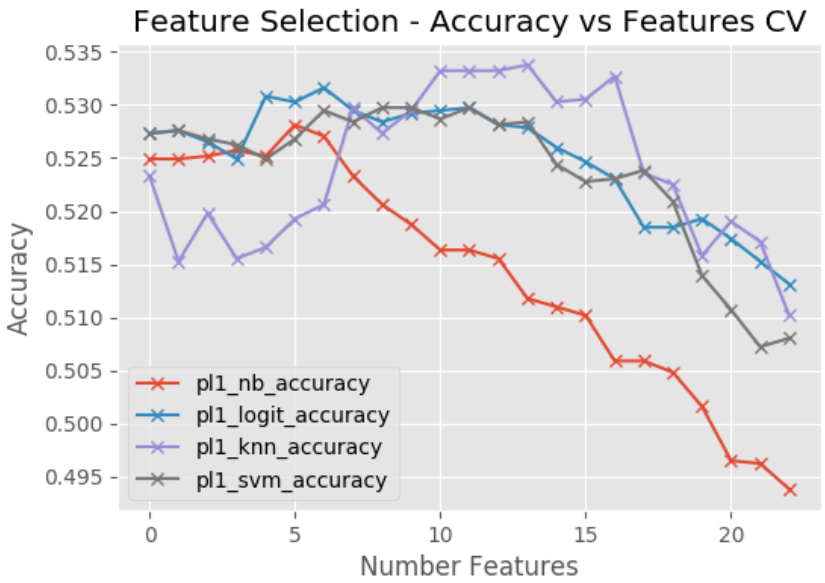

Figure 3: Forward Selection - Platinum

Exhaustive Selection

Another method for gaining some insight into the optimal subset of features is what might be called exhaustive selection. This method considers using one feature of each type - return, momentum, sma, volatility - together with the three categorical features - sr, im, hlm - over all possible choices. As there are 5 choices for each of the 4 groups of feature that we will vary, this leads to $5^4 = 625$ possible combinations of features. The same average accuracy metric as above is used for comparing between the groups.

Figure 4 below shows how the accuracy varies by feature combination. It is interesting to note that the kNN method is clearly more sensitive to the collection of input features than the other classifiers. We also see 'periodic' behaviour indicating that certain classifiers do perform noticeably better on this training set with certain values of the predictors. Table 2 highlights the particular case of Momentum. The figures here are averages over accuracy for all combinations of features tested with the relevant value of momentum. We see that the 1d momentum feature results in more accurate models for logistic regression and the support vector classifier than the 5d momentum feature. Although the differences appear small on an absolute scale, they may be quite substantial when considered in the context of the problem and without aggregation.

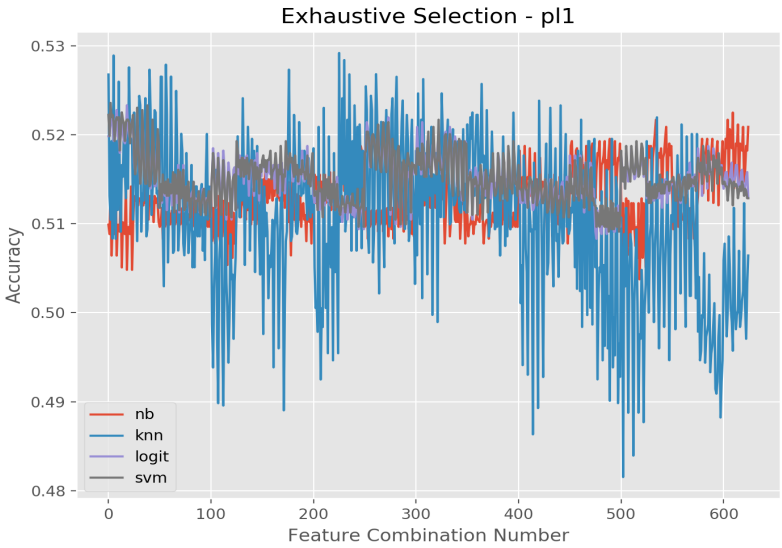

Figure 4: Exhaustive Selection - Platinum

Momentum	NB	kNN	Logit	SVM
1d	51.04%	51.21%	51.74%	51.76%
2d	51.41%	51.08%	51.50%	51.55%
3d	51.17%	51.31%	51.51%	51.49%
4d	51.40%	50.80%	51.49%	51.48%
5d	51.43%	50.83%	51.30%	51.30%

Table 2: Exhaustive Selection - Platinum, Momenum

Results

All results in this section were obtained through use of sklearn's GridSearchCV method. For each classifier I computed three models - one using all features, one using the top N features discovered through the forward selection procedure, where N is the high point in terms of accuracy, and one discovered through the exhaustive selection procedure. No consideration has been given to temporal structure here i.e. the training and testing sets will contain data from each year. Features were rescaled before fitting for all classifiers.

Denote the classifiers by NB, kNN, SVC, LR with the obvious meanings. The search space for each model was as follows:

- NB: $\alpha = [0.1, 0.2, \ldots, 1.0]$

- kNN: neighbours = $[1, 3, \ldots, 29]$, metric = ['manhattan', 'euclidean']
- SVC: penalty = $[L^1, L^2]$, $C = [100, 10, 1, 0.1, 0.01]$
- LR: penalty = $[none, L^1, L^2]$, $C = [100, 10, 1, 0.1, 0.01]$

The optimal collections of parameters chosen for the above were:

- NB: $\alpha = 0.1$
- kNN: neighbours = 5, metric = manhattan
- SVC: penalty = L^1, $C = 0.01$
- LR: penalty = L^1, $C = 0.1$

Tables 3 and 4 display performance figures and the relative types of classification error equivalent to a confusion matrix i.e. True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN). In all performance categories, the ordering of best to worst was NB, SVC, LR, kNN. We see that all models had more false negatives than they did true negatives. We will come back to this in the section on Prediction Quality.

	accuracy %	precision %	recall %	f1 %
NB	53.47	57.21	53.52	55.31
kNN	50.56	54.47	49.33	51.77
SVC	52.98	56.67	53.52	55.05
LR	51.94	55.95	50.07	52.85

Table 3: Copper Model 3 Performance

	TP	FP	TN	FN
NB	357	267	306	310
kNN	329	275	298	338
SVC	357	273	300	310
LR	334	263	310	333

Table 4: Copper Model 3 Classification Statistics

Logistic Classifier

Hyperparameters to be tuned here are the form of the turnover penalty and C, which determines the (inverse) regularisation strength. In more detail, fitting a logistic regression is equivalent to solving the following optimisation problem:

$$\min_w \frac{1}{p}||w||_p^p + C\sum_{i=1}^{m} \log\left(1 + e^{-y_i\langle x_i, w\rangle}\right),$$

where $p = 1, 2$ is the choice of regularisation penalty, $y_i \in \{-1, 1\}$ is the class of observation i, $x_i \in \mathbb{R}^n$ is the i-th sample consisting of n feature measurements and $w \in \mathbb{R}^n$ is the weight or coefficient vector to be solved for.

It is clear from this framing of the problem that large values of C give relatively less importance to the regularisation term. We should therefore expect that small values of C will reduce the size of the coefficients for both $p = 1$ and $p = 2$. Only with the lasso, or L^1 penalty, should we expect the sparsity property. This is the property of coefficients being forced to zero as C decreases leading to a 'sparse' model in terms of features.

Table 5 displays the coefficients resulting from using no penalty, together with the L^1 and L^2 penalties with $C = 10$ and $C = 0.1$. As we would expect, decreasing C increases the impact of the penalty term, shrinking the values of the coefficients and forcing increasingly many to be zero in the case of the L^1 penalty. Note that these models were trained on the full set of 23 features and we see a trend of increased accuracy as models become more heavily regularised. This may imply that using the full set of features causes overfitting.

Name	Accuracy	Non-Zero Coeff.	Avg. Abs Coeff.	Max Coeff.
No Penalty	50.16%	23	0.073	0.355
L^2, $C = 10$	50.32%	23	0.079	0.378
L^2, $C = 1$	50.16%	23	0.0738	0.355
L^2, $C = 0.1$	51.45%	23	0.048	0.232
L^1, $C = 10$	50.24%	20	0.071	0.371
L^1, $C = 1$	50.73%	16	0.049	0.288
L^1, $C = 0.1$	52.02%	8	0.014	0.044

Table 5: Impact of Regularisation on Logistic Regression

Support Vector Classifier

Recall the distinction between a (hard) maximal margin classifier and a (soft) support vector classifier. In the case of a maximal margin classifier, we must assume that the underlying feature space is linearly separable i.e. there exists a hyperplane $\Pi := \{x \in \mathbb{R}^n | \, b + \langle x, w \rangle = 0\}$ such that $y_i(b + \langle x_i, w \rangle) > 0$, for $i = 1, \ldots, m$ where $y_i \in \{-1, 1\}$ is the i-th class label. Such a plane can be parameterised such that the margins are represented by $b + \langle x^{\pm}, w \rangle = \pm 1$ and hence $\langle x^+ - x^-, w \rangle = 2$. Since w and $x^+ - x^-$ are normal to the plane, it follows that $2 = ||w||_2 \, ||x^+ - x^-||_2 \cos(0)$. Therefore, the maximal margin separating hyperplane will be a solution to the optimisation problem:

$$\max_{b,w} \frac{2}{||w||_2} \text{ s.t } y_i(b + \langle x_i, w \rangle) \geq 1, \; i = 1, \ldots, m.$$

This is equivalent to the following minimisation problem:

$$\min_{b,w} \frac{1}{2} \langle w, w \rangle \text{ s.t } y_i(b + \langle x_i, w \rangle) \geq 1, \ i = 1, \ldots, m.$$

Finally, in the case that the data is no longer linearly separable, we can make allowances for points to cross the margin leading to the so-called soft margin classifier. In a situation as complex as predicting stock price direction, it seems unlikely that any useful, separable feature spaces will be found. We can quantify the amount by which a point breaches the margin by $\max(0, 1 - y_i(b + \langle x_i, w \rangle))$. With this in mind, the optimisation problem can then be reformulated as it is displayed in the sklearn documentation:

$$\begin{aligned} &\min_{b,w} \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^{m} \zeta_i \\ &\text{s.t } y_i(b + \langle x_i, w \rangle) \geq 1 - \zeta_i, \ \zeta_i \geq 0, \ i = 1, \ldots, m. \end{aligned}$$

For the purpose of illustration, figure 5 shows a SVC run on the rescaled copper features 1d return and 1d momentum, only, with $C = 1$. The training data is not close to being separable and, indeed, almost 94% of sample vectors are included in the collection of support vectors. This figure highlights the fact that this commodity is mean reverting, i.e. an up more is more likely following a down move and a down is more likely following an up move, and the model has captured this behaviour.

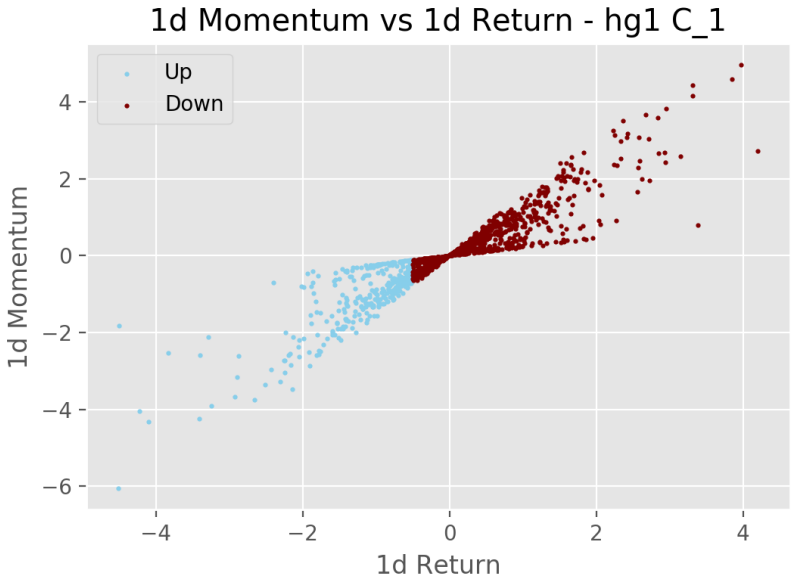

Figure 5: SVC - Copper 1d Momentum vs 1d Return

K-Nearest Neighbour Classifier

Classification with the kNN supervised learning algorithm is often described as lazy as no model training takes place before a new point is to be classified - the model is in a sense generalised only after the new point has been classified. I will not revisit the choice of the optimal number of neighbours to use here as this was discussed in the results section.

Plots 6 and 7 below show the decision boundaries produced for Platinum when using a kNN classifier trained only on 20d volatility and 1d return using $k = 5, 100$ neighbours, respectively. It is interesting to note that the boundary shifts almost completely based on the number of neighbours used for classification in this example.

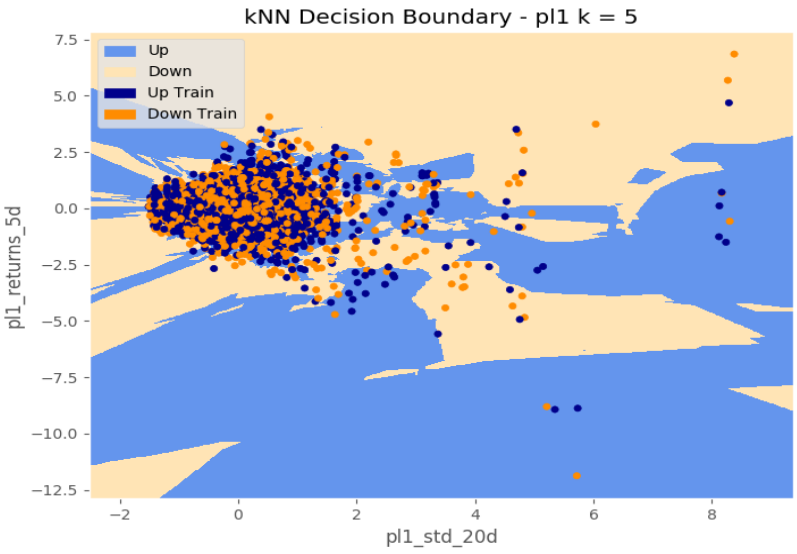

Figure 6: kNN - 1d Return vs 20d volatility, $k = 100$

Prediction Quality & Bias

Unfortunately, none of the models trained in this study provides particularly strong performance, although we are able to slightly improve upon the benchmark predictor. Figure 8 below shows the ROC curves for copper that were fit in the Results section above using a 5-fold cross-validation grid search with 25% of data held out for testing. The hold-out test data was then used to plot these curves. As the LinearSVC class of sklearn does not support probability computations, the relevant ROC curve was computed using the more general

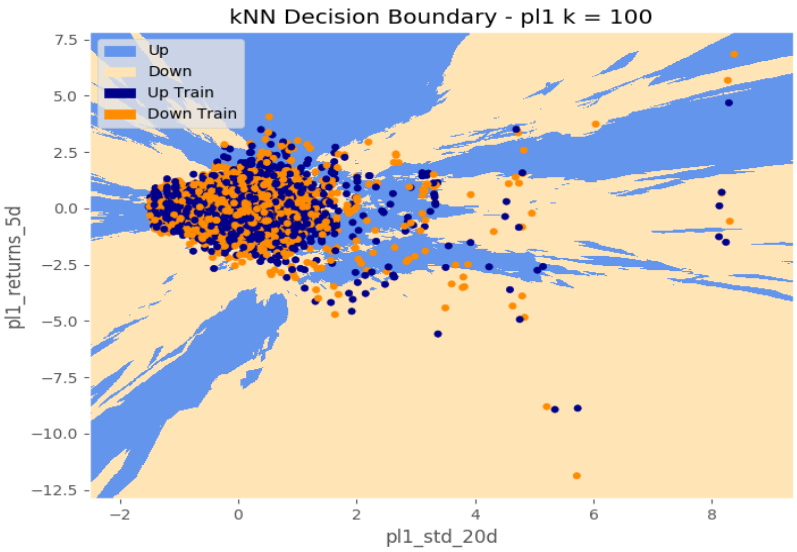

Figure 7: kNN - 1d Return vs 20d volatility, $k = 100$

SVC class with a linear kernel and the same C value as found to be optimal with the grid search.

The areas under each curve are contained within table 6. We see that the naive Bayes classifier is the best under this measure with logistic regression coming second although there is not a major difference between the models, all of which are just slightly better than random. Combining this result with the figures in table 3 we can conclude that naive Bayes performed best on this data set overall for copper. I believe the drop of SVC in the hierarchy when compared to earlier results is due to differences between the implementations of SVC and LinearSVC. However, since probabilities will be required for the backtesting section, I will use logistic regression as the second-best model going forward.

	NB	kNN	SVC	LR
ROC Area	53.16%	52.49%	52.33%	52.57%

Table 6: Area Under ROC Curve

Negative Return Prediction

In this section I will briefly discuss the prediction performance for down days, or negative returns, as mentioned previously. I have not included the results

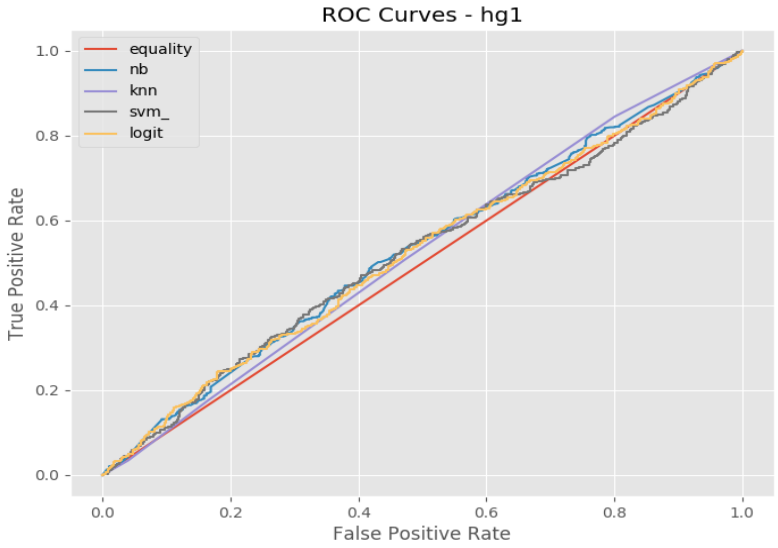

Figure 8: Receiver Operator Characteristic Curves, model 3

in this report (they are available in the submission) for platinum, but there we see many models predicting almost exclusively positive moves, which can lead to high recall and even $f1$ scores. These models, however, may not form a particularly useful basis for developing a trading strategy. Indeed, in most markets we would expect the most violent moves to be on the downside and so such predictor may naturally give rise to highly negatively skewed performance profiles.

From table 4, we see that each model has more false negatives than true negatives. This may be related to the fact that up moves have been slightly more likely over the course of the training period. One way to reduce the number of false negatives would be to reduce the threshold required for an up prediction. Instead of requiring our probability model to indicate a more than 50% chance of an up move, we may only require a threshold of 48%, say, before we predict such a move. The optimal level of threshold could be tested through cross-validation.

	NB	kNN	SVC	LR
$\mathbb{P}(N \vert predict\, N)$	49.68%	46.86%	49.18%	48.21%

Table 7: Probability Negative Given Negative Prediction

Transition Probabilities and Backtesting

I will use the NB and LR models outlined in the previous sections to backtest the trading strategy outlined in the brief. Betting $2p - 1$ very quickly resulted in bankruptcy for the majority of backtests I ran and so I introduced a scale factor to reduce volatility. Figure 9 below shows the live strategy performance for a logistic regression classifier that was trained over a 3 year period and then run live from 30/04/2019 - 30/04/2020. This example is fairly representative of the kinds of results I obtained i.e. mostly a NAV decaying toward zero. In particular, there are several days where the classifier was up to 70% confident in the next day's direction. Such predictions can prove very destructive.

Several possible improvements to strategy, aside from training a better predictor, might be:

- Scale positions inversely to asset volatility, thereby reducing the overall volatility
- Only trade when the prediction probability has breached a threshold. For example, $p > 52\%$ for a buy and $p < 48\%$ for a sell. Here p is the probability assigned by the model to an up move. This thresholds need not be symmetrical and could also vary with market volatility
- Trade several assets together to benefit from portfolio diversification

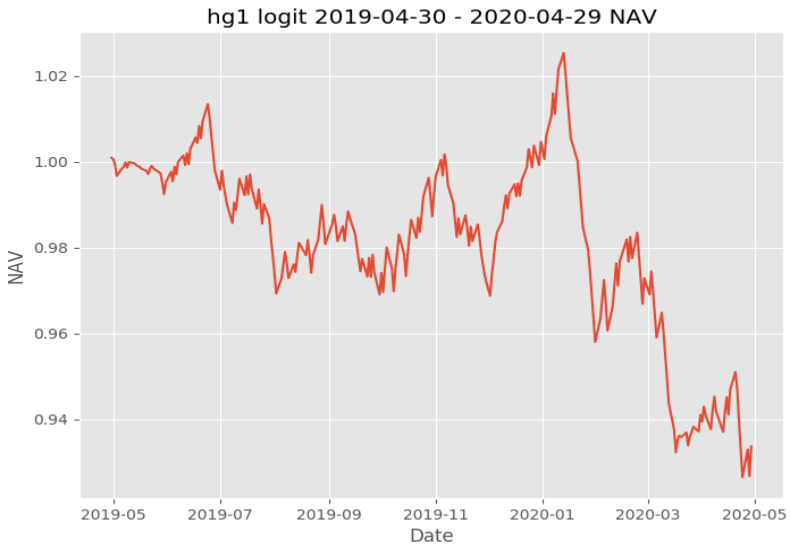

Figure 9: Copper Logit Backtest - Scale Factor 10%

Exponential Family - Normal Distribution

Treating the variance σ^2 as known, we aim to write:

$$f(x;\mu) := \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)} = e^{a(x)b(\mu)+c(\mu)+d(x)}$$

Taking logarithms of both sides and expanding the quadratic term:

$$\frac{x\mu}{\sigma^2} - \frac{\mu^2}{2\sigma^2} - \left(\frac{x^2}{2\sigma^2} + \log(\sigma\sqrt{2\pi})\right) = a(x)b(\mu) + c(\mu) + d(x)$$

One possible choice for the functions is then:

$$\begin{aligned} &a(x) = x/\sigma,\ b(\mu) = \mu/\sigma \\ &c(\mu) = -\mu^2/(2\sigma^2) \\ &d(x) = -(x^2/(2\sigma^2) + \log(\sigma\sqrt{2\pi})) \end{aligned}$$

References

[1] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.