Machine Learning Engineer Nanodegree

Capstone Project – Use clustering and multiple Tabular Predictors to predict the 1-day future price of gold

Neil Simon

January 14, 2022

I. Definition

Project Overview

Predicting changes in the value of gold has obvious financial benefit including knowing when to buy or sell and determining future volatility. Historically, gold has been used as a currency, either directly or as backing for a paper currency (Gold's history as a currency standard, 2010) and it has proven to be a useful asset to hold during times of economic downturn and uncertainty. This is despite the relatively small role it plays as a functional material in the modern world. The efficient market hypothesis (EMH) suggests that the market price reflects all information and that it should be impossible to consistently predict future prices of gold, and in particular, that it is impossible to consistently get returns greater than market level returns (Fama, E. F., 1970). Therefore, any system which is able to beat the market consistently would not only provide significant avenues for financial return, but also demonstrate that the EMH is not perfectly supported. In this project, I implemented a system which divides gold prices into clusters using a K-Means predictor and related historical data, and based on the predicted cluster, uses a Tabular Predictor trained for that cluster to determine a predicted 1 day future return on gold. Having analysed the results, and compared them with naive trading strategies, I have determined that none of the configurations resulted in a system capable of predicting the future value of gold with any significant degree of accuracy.

Problem Statement

The problem I attempted to solve is to predict the next day's gold price based on historical gold price trends and the Standard and Poors 500 index (S&P 500). To do this I performed the following tasks:

1. Retrieve the historical price of gold and historical values of the S&P 500.
2. Combine such into a suitable table of data.
3. Add additional columns of data expressing historical and most recent returns for gold and S&P 500.
4. Train a K-Means clustering system to predict clusters.
5. Train K different Tabular Predictors on data based on predicted cluster of said data.
6. Gather predictions on test data.
7. Compare results to actual returns on predicted days.

8. Iterate with different numbers for K (different numbers of clusters).
9. Evaluate final results.

By comparing predicted return and therefore price of gold with the actual market price I determined how accurately the model predicts such and the associated level of risk. Additionally, based on these predictions I extrapolated an expected return of the model over a long period and compare such with the actual market return (always buy/hold strategy) and with other naive trading strategies.

Metrics

The metrics I have applied in evaluating the predictions are relative cumulative return based on recommended trading, and accuracy. To evaluate the cumulative return over the test data and compare such with the actual return. The return for a day can be calculated thus:

$$return = \frac{closing\_price}{opening\_price} - 1$$

The cumulative return is just a matter of getting the product of all actual returns:

$$c\_return = (1 + return_1) * (1 + return_2) * ... * (1 + return_{n-1}) * (1 + return_n)$$

Of course, if we know that the return is going to be positive (gold going up in value), it makes sense to buy or hold, and if it is going down, we sell. Therefore we can bring the concept of placing a buy or sell order and the ideal order is thus:

$$order_{ideal} = \begin{cases} +1 & \text{if } return \geq 0 \\ -1 & \text{if } return < 0 \end{cases}$$

Thus we can change the above to give an idealised return:

$$c\_return_{ideal} = (1 + order_1 * return_1) * (1 + order_2 * return_2) * ...$$

Assuming that the predicted values are the ideal, we can use the above and create a formula for $c\_return_{recommended}$ where we follow the recommendations of the system and use the following order formula:

$$order_{recommended} = \begin{cases} +1 & \text{if } return_{predicted} \geq 0 \\ -1 & \text{if } return_{predicted} < 0 \end{cases}$$

Thus the actual cumulative return, if following this strategy of trading recommended by the predicted values, is:

$$c\_return_{recommended} = (1 + order_1 * return_1) * (1 + order_2 * return_2) * ...$$

Thus we can compare the return that following the recommended system would give us, the return that simply buying/holding gold would (the market return) and a fair random coin (an average cumulative return of 1). For comparing returns, it makes most sense to consider the market as the baseline and view the relative cumulative return of the recommendation system:

$$r\_return_{recommended} = \frac{c\_return_{recommended}}{c\_return}$$

A value greater than 1 would suggest a system that has outperformed the actual market.

As it is possible to outperform the market through random chance and since the cumulative return is a product of all the predictions, I also compare the $order_{ideal}$ and $order_{recommended}$ for accuracy to see if the system does a good job of accurately predicting the direction of gold prices.

$$accuracy = \frac{count(order_{ideal} == order_{recommended})}{number\_of\_samples}$$

A value consistently greater than 0.5 would indicate a greater degree of accuracy than pure chance.

Additionally, to check for bias, I calculated precision, recall and F_1 score.

$$precision = \frac{true\_positives}{true\_positives + false\_positives}$$

$$recall = \frac{true\_positives}{true\_positives + false\_negatives}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

II. Analysis

Data Exploration

I have used a combination of 2 different datasets, one is the price of gold since 1994-12-06 and 2021-12-06, and the other the closing price of the S&P 500 between 2006-01-04 and 2021-12-03. These datasets were retrieved from:

1. Gold price: https://www.kaggle.com/nward7/gold-historical-datasets
2. S&P 500 Historical Data: https://ca.investing.com/indices/us-spx-500-historical-data

The gold price dataset included the following fields:

1. Date: in MMM DD, YYYY format
2. Price: Closing price in USD

3. Open: Opening price in USD
4. High: Daily high price in USD
5. Low: Daily low price in USD
6. Vol: Volume of traded gold
7. Change %: Daily return - not actually a percentage, but decimal fraction

The S&P 500 dataset included the following fields:

1. Date: in MMM DD, YYYY format
2. Price: Closing price in USD
3. Open: Opening price in USD
4. High: Daily high price in USD
5. Low: Daily low price in USD
6. Vol: Volume of traded - no values present
7. Change %: Daily return

As we can see, these datasets have many of the same fields, but for the purposes of the remainder of this project, I only used the date and closing price fields from these two datasets. By combining these two I created a third dataset with the date, gold closing price and S&P 500 closing price. The resulting dataset has 4008 records for trading days, starting at 2006-01-04. This resulted in a dataset with 3 fields:

1. Date
2. Gold Price
3. S&P Price

Exploratory Visualization

For the purposes of visualisation, I created a chart (see Figure 1) showing the closing prices relative to the price on the first day. From this we can see that the closing prices do have some degree of relationship, but not necessarily a very close one.

I further decided to create a chart (see Figure 2) showing the price relative to the price at halfway through the dataset. This chart shows a closer relationship. We can see a similar slope, but that when the S&P 500 market deviates down from that slope, the gold price increases, and vice versa. This suggests that gold is being treated as a safe haven for investors when the S&P 500 is performing poorly, and the S&P 500 is generally seen as a good indicator of the stock market in general. This in fact is the general consensus (Chen, 2021).

It is hoped that this feature of the price relationship between gold and the S&P 500 will provide information on the future price of gold.

Algorithms and Techniques

I decided to add further information using derived features for each day's data to add 16 days worth of historical information to that day's data, and in so doing

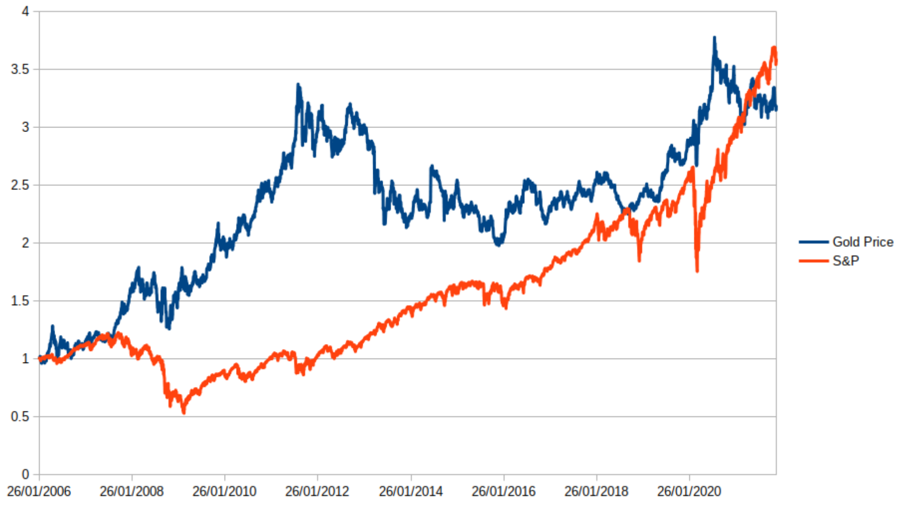

Figure 1: Gold and S&P 500 Return

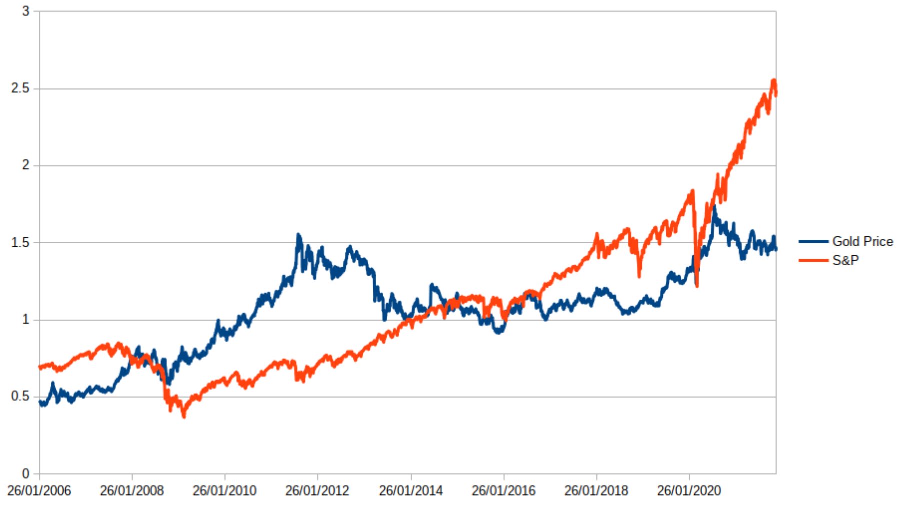

Figure 2: Gold and S&P 500 Return Relative to Midpoint

intended to provide relevant technical information about the historical price of gold and the S&P 500 over the preceding days.

I decided to use a K-Means clustering algorithm to separate days into clusters based on the additional historical data added to each day's dataset. This was done to try to find days with similar historical trading returns, and cluster them together. Essentially, this was hoped to group certain "shapes" of trading patterns together. For each cluster, I trained a Tabular Predictor with only days that belonged to that cluster. By doing so, I hoped to have Tabular Predictors which were experts in specific historical trading patterns and ultimately give better results than training as a single whole. Figure 3 is a simplified flow chart of this system.

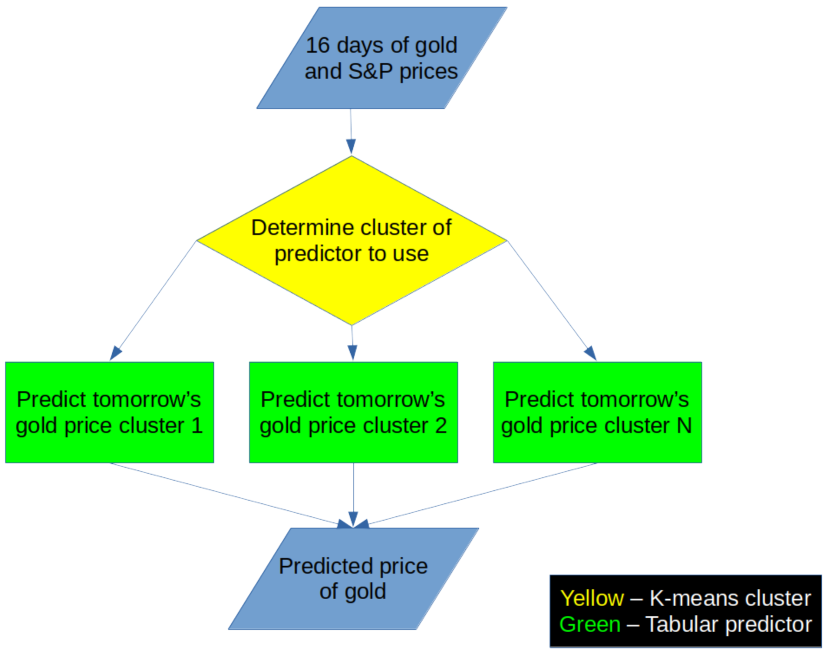
16 days of gold and S&P prices
Determine cluster of predictor to use
Predict tomorrow's gold price cluster 1
Predict tomorrow's gold price cluster 2
Predict tomorrow's gold price cluster N
Predicted price of gold
Yellow – K-means cluster
Green – Tabular predictor

Figure 3: Model of System

There are multiple ways to alter/tune this system:

- The number of clusters K.
- The additional historical information added to each day's data.
- Add other additional information such as other market historical information, such as the FTSE 100, the DAX or the Dow Jones.

For the purposes of this project, I only varied the number of clusters, using the values 1, 2, 4, 8, and 16. When using a single cluster, it was effectively equivalent to not using the K-Means clusteriser and thus unclustered.

Benchmark

The benchmarks against which I measured this system were:

- The average return of a fair coin flip.
- The market return.

These benchmarks are all possible to derive trivially from the historical data.

The average return of a fair coin flip is 1 as the daily return is either positive or negative by the same proportion and with equal likelihood. The market return is easily gathered by just using the cumulative return as described in Metrics. Similarly, the ideal return is also described in the Metrics above.

There are no publicly available models which perform significantly better than market return, so there are no other models against which it makes sense to compare.

III. Methodology

Data Preprocessing

1. Add additional historical data to each day's data.
2. Normalise this data using the variance.
3. Verify that the data is suitably normally distributed.
4. Split the data into training and testing sets.

From this dataset I generated the gold return for the ranges of 1 day's return, 2 day's return ending yesterday, 4 day's return ending 3 days back, and 8 day's return ending 7 days back. The general formula for this is:

$$return_{(x,y)} = \frac{p_{n-x} - p_{n-x-y}}{p_{n-x-y}}$$

Where p_n is today's gold price, and p_{n-1} is yesterday's gold price, etc., x is the number of days back we are looking to find the return at, and y is the number of days over which the return is calculated. So for today's return (1 day's return ending today) $x = 0$ and $y = 1$, and for 4 day's return ending 3 days back $x = 3$ and $y = 4$.

I then normalised these features by getting the 16 day standard deviation on daily returns and dividing the returns by that (adjusted for the number of days over which the return was calculated). The normalization formula for these is:

$$return\_norm_{xy} = \frac{r_{xy}}{\sigma_{16} * \sqrt{2}y}$$

This resulted in 4 normalised return values for each day's gold price and a standard deviation value.

I applied the same approach to the S&P 500 data to create 4 normalised return values for each day, and a standard deviation value. Resulting in a total of 10 features with information for each day going back 16 days of trading. I added these additional features to each row to increase the amount of information about recent gold and S&P 500 returns, while not massively increasing the total amount of information in the dataset. While it would be possible to have just included the individual daily returns going back multiple days in each row, this would have increased the degrees over which the K-Means clustering operated, and therefore resulted in a less useful clustering process. My approach was to attempt to maximise the amount of information in each row, without increasing the number of features too greatly, so as to potentially benefit from the use of the K-Means clustering algorithm.

Ultimately, it is the one day future return ($\overline{return}_{(-1,1)}$) we are looking to calculate as from that we can determine tomorrow's price as we know today's price. So that was added as a label to the data for use in training and evaluation of the model.

After creating the normalised returns, I examined the data to make sure that it was somewhat close to normally distributed. While each does display some degree of variance from the expected normal distribution, it is sufficiently close to be assumed that it is normally distributed and relatively anomaly free for the purposes of this project (see Figure 4 through 11).

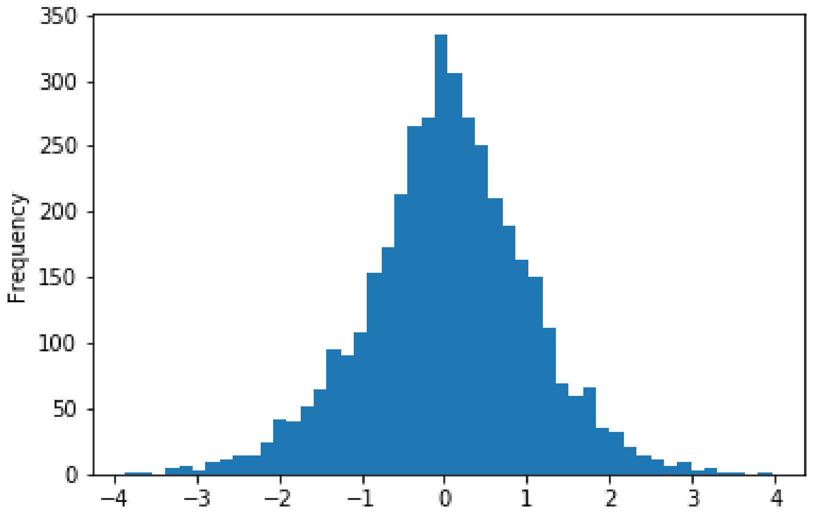

Figure 4: Normalised Gold Return Over 1 Day

Additionally, this data was split into two sets of data for training and testing.

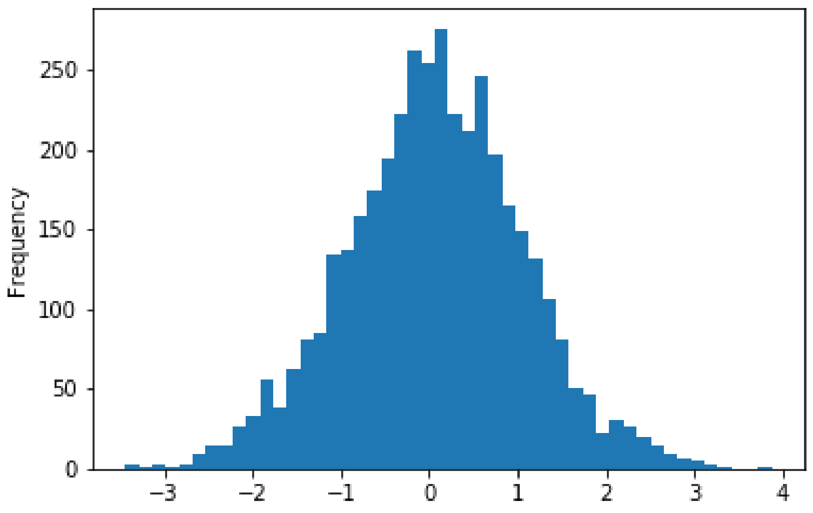

Figure 5: Normalised Gold Return Over 2 Days Ending 1 Day Ago

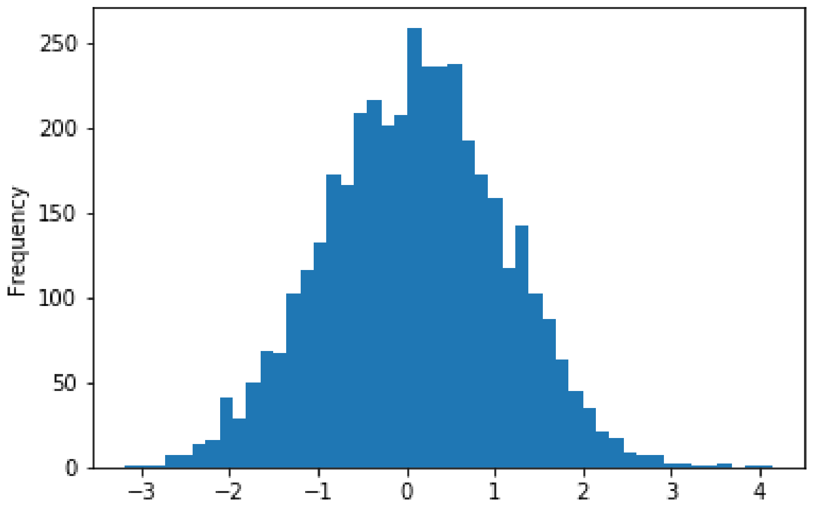

Figure 6: Normalised Gold Return Over 4 Days Ending 3 Days Ago

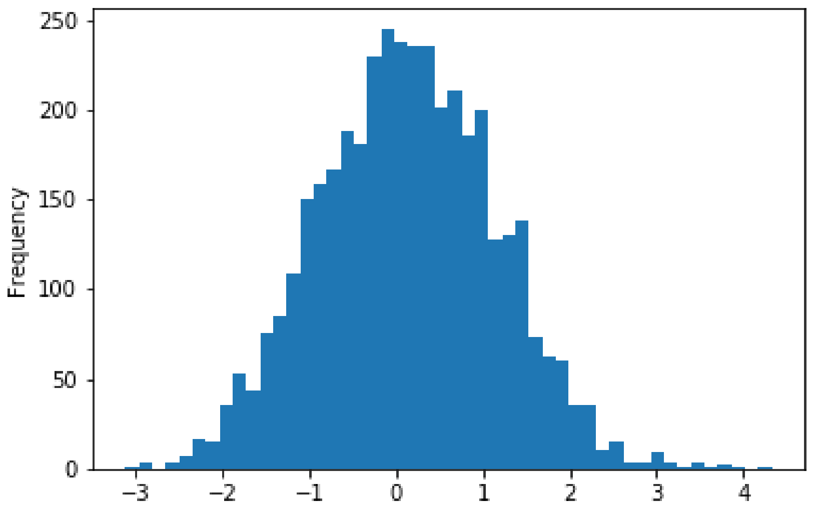

Figure 7: Normalised Gold Return Over 8 Days Ending 7 Days Ago

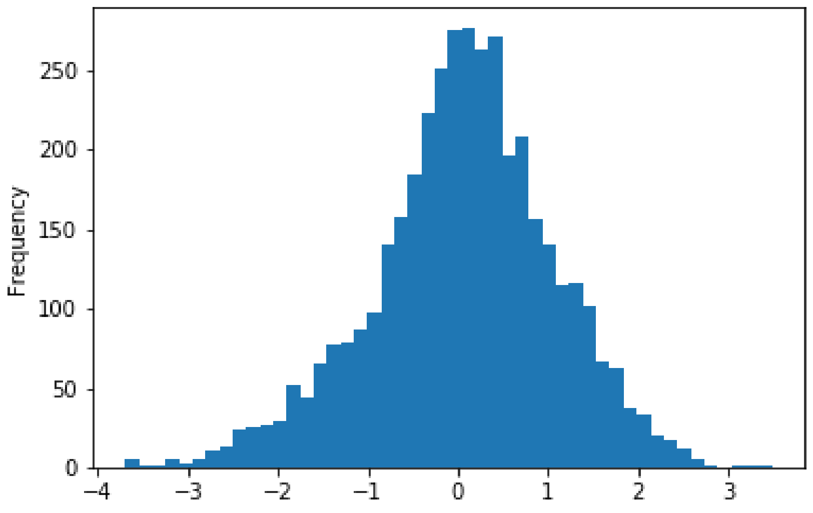

Figure 8: Normalised S&P Return Over 1 Day

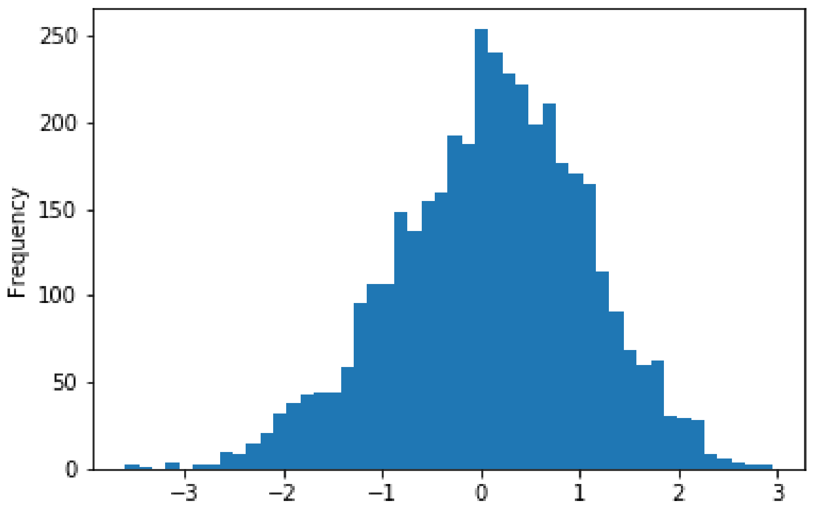

Figure 9: Normalised S&P Return Over 2 Days Ending 1 Day Ago

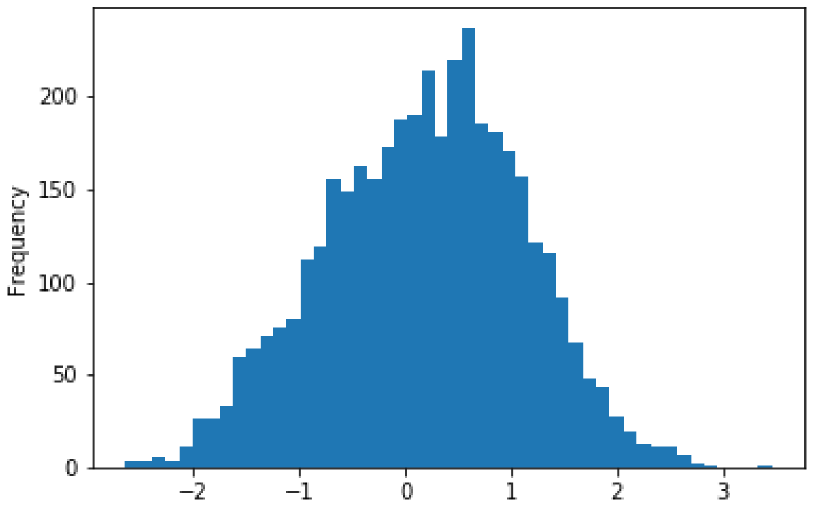

Figure 10: Normalised S&P Return Over 4 Days Ending 3 Days Ago

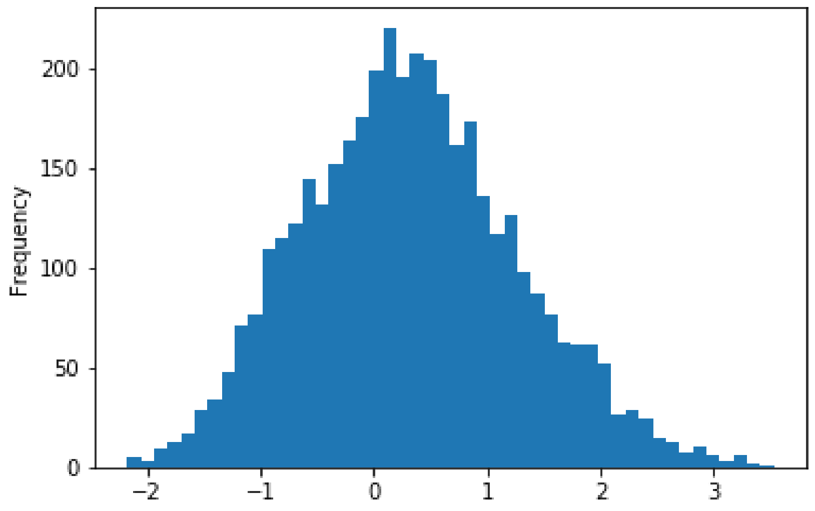

Figure 11: Normalised S&P Return Over 8 Days Ending 7 Days Ago

The training set I chose was the first 3000 days of trading and the testing set was the last 992 days. The reason for not randomising this data before splitting into training and testing sets is that in doing so we could potentially introduce information about the future in the training data relative to the testing data, giving it a degree of knowledge of what trading patterns would be like in future relative to the data on which it was testing. Such would entirely defeat the purpose of this system as we do not and cannot train a production system on data set in the future.

Implementation

To implement the above approach, I used an AWS Sagemaker Jupyter Notebook (https://github.com/neilsimon/nd009t-capstone/blob/main/Capstone%20Training.ipynb). The initial step required was to pull the pre-processed data as was to be used into the notebook as a Pandas dataframe.

I then split this data into training and testing data with 3000 days in the former and 992 in the latter.

For the purposes of the K-Means algorithm, I created versions of the data with the required features:

- Normalised Gold Return
- Normalised Gold Return 2 day back 1
- Normalised Gold Return 4 day back 3

- Normalised Gold Return 8 day back 7
- Var 16 day (gold)
- Normalised S&P Return
- Normalised S&P Return 2 day back 1
- Normalised S&P Return 4 day back 3
- Normalised S&P Return 8 day back 7
- Var 16 day.1 (S&P 500)

This resulted in 10 features upon which the clustering algorithm would work.

To create a K-Means endpoint, I used the SageMaker library K-Means and trained it on the training set for the desired number of clusters (K initially set to 4). With the resulting model, I deployed a predictor endpoint and used that to get the clusters for each of the training and testing rows. I added this additional cluster information to the training and testing data for use with the Tabular Predictor.

To train the Tabular Predictors, I used an AutoGluon auto-ml setup derived from the AutoGluon SageMaker tutorials (Cloud Training with AWS SageMaker, n.d) and (Deploying AutoGluon Models with AWS SageMaker, n.d). These referenced a code repository from which the remaining code for training and deploying Tabular Predictors is derived (https://github.com/aws/amazon-sagemaker-examples/tree/master/advanced_functionality/autogluon-tabular-containers). The script was significantly modified to make it suitable for both training and predicting (https://github.com/neilsimon/nd009t-capstone/blob/main/scripts/hpo.py).

I split the training and testing sets into K different sets of each, and trained an AutoGluon Tabular Predictor on each of these clustered sets. Afterwards, I gathered the predictions for each cluster divided testing set and combined them into a single, date sorted prediction table, which I had written to a spreadsheet for further analysis.

Additionally, the time taken for the run from start to finish was recorded. This was done to determine how many clusters was the largest which I could reasonably deploy.

The most complicated part of deploying this implementation was finding a way to deploy the AutoGluon Tabular Predictor and verify that the setup for doing so was working correctly.

Refinement

The largest refinement I undertook was to vary the number of clusters used. I had initially expected to just have the resources to do so for 4 clusters, but ultimately after I had everything working it was easy to add other cluster sizes and so I tried 1, 2, 4, 8, 16 and even 32. It was this last one that proved to be a problem as the AWS system did not permit my account to deploy 33 endpoints, as would be needed, and so the run crashed, leaving 30 endpoints deployed and ultimately consuming the remaining resources available.

Additionally, I changed the configuration used for training the Tabular Predictor after the first run as it was very quick in training using a "medium_quality_faster_train" preset (https://github.com/neilsimon/nd009t-capstone/blob/main/ag_configs/config-med.yaml), so moving to the "best_quality" preset (https://github.com/neilsimon/nd009t-capstone/blob/main/ag_configs/config-full.yaml) incurred only a small relative overall hit in processing time.

IV. Results

Model Evaluation and Validation

The 5 successful runs of the system, with K sizes 1, 2, 4, 8, and 16 were compared with the market return, and fair coin flip return. The following results were achieved:

Configuration	C. Return	R. Return	Accuracy	Precision	Recall	F_1	RMSE	Train(s)
Unclustered	0.813	0.627	0.467	0.530	0.130	0.209	0.010	1255
2 Clusters	0.978	0.754	0.485	0.543	0.305	0.391	0.027	1933
4 Clusters	1.191	0.919	0.500	0.537	0.549	0.543	0.038	2614
8 Clusters	0.924	0.713	0.487	0.534	0.406	0.461	0.049	4105
16 Clusters	1.869	1.442	0.523	0.572	0.473	0.518	0.148	7653
Coin Flip	1.000	0.772	0.500	0.541	0.500	0.520		
Market Return	1.296	1.000	0.541	0.541	1.000	0.702		
Always Sell	0.706	0.545	0.459	0.000	0.000	0.000		

Based on these results, were we implementing a long term production version of this tool, we would use the 16 Clusters configuration as seems to give a better return, and has higher level of accuracy and precision, indicating that it is a better fit. That said, the comparatively high value for the root-mean-square error (RMSE) does indicate a relatively high level of deviation from the actual daily return and therefore a relatively high risk.

Figure 12 shows the above table in chart form. Figure 13 shows the relationship between cluster count, and training and deploy time.

Justification

As we can see, the clustered approach as described above did not achieve particularly useful results, generally failing to beat the always buy/hold and not even consistently achieving better results than the coin flip approach to trading. The 2 configurations that did achieve better results than the market (the always buy approach) were the 4 cluster and 16 cluster approaches. I suspect that this was more due to random chance than anything else, but the 16 Cluster

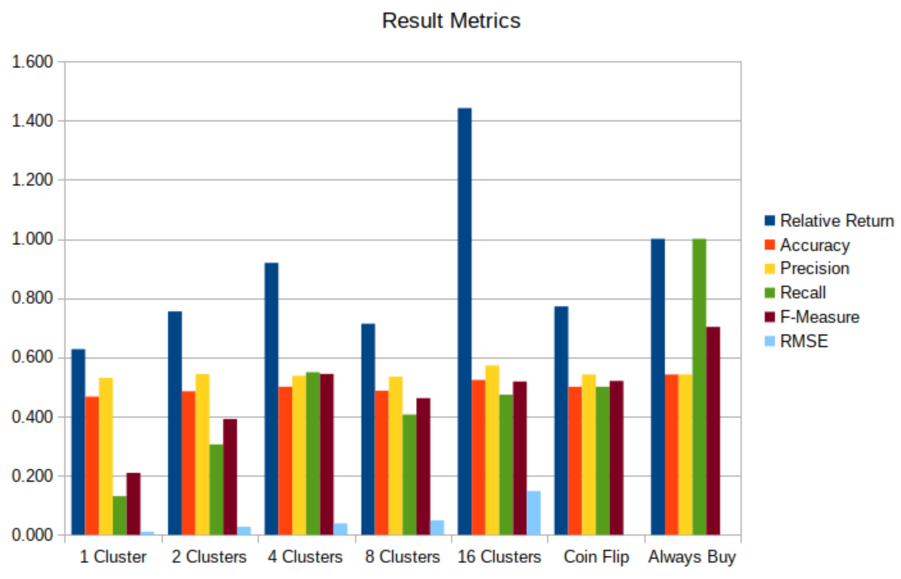

Figure 12: Results

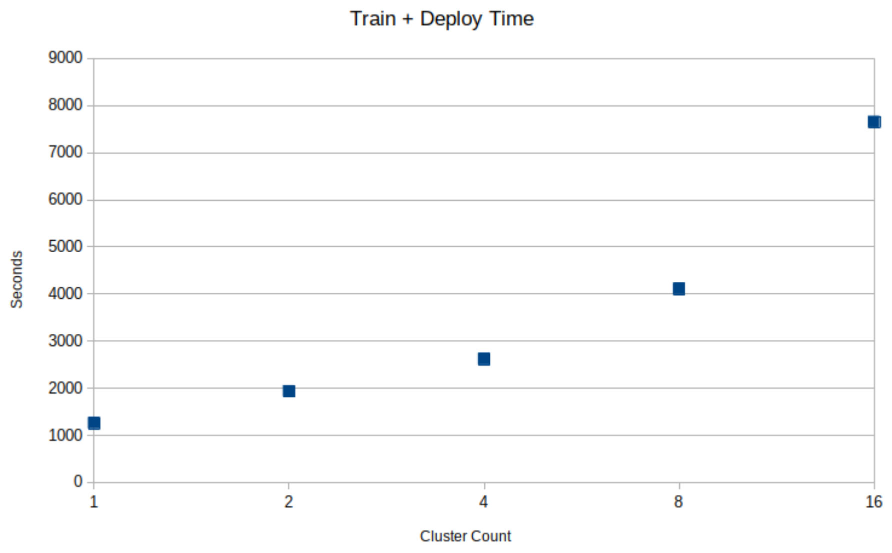

Figure 13: Training and Deploy Time

configuration does show some promise. Its particularly high relative return, and accuracy suggest that it may be possible to use it to beat the market over the long term. That said, the data required to test this does not exist as we would need a large number of values to have any significant degree of certainty.

The low values for recall and F_1, combined with the relatively high RMSE suggest that the risk associated with the 16 cluster configuration is rather high. In particular, the relatively high RMSE when compared with configurations with fewer Tabular Predictor systems (fewer clusters, or a single cluster/unclustered) suggests that the amount of data used for the Tabular Predictor training was too small. The average size of such a training set would have been 188 rows, which is rather small and would lead to an underfit model. Unfortunately there is no way to add more data for training, as any such data would have to extend further back in time and would likely be less relevant to current trading.

Ultimately, none of these configurations is expected to beat the strategy of always buying and holding gold as they provide either too much risk or a lower relative return.

V. Conclusion

Free-Form Visualization

As we can see (Figure 14) when we apply the 16 cluster configuration to the test data, it does give an above market return, generally following the market itself but usually doing better. Interestingly though, it does for a period actually drop below the market return, so it cannot be considered to give consistently good predictions. The relatively chaotic nature of the market and the fact that return is multiplicative, means that even a relatively good performing system may well underperform the market over significant time frames, and similarly that relatively low accuracy systems might even outperform the market over such too.

Reflection

The initial problem was to find a potentially novel approach to predict gold values, and to ultimately determine if such a system could show some weakness in the Efficient Market Hypothesis. While I considered a variety of possible solutions, I decided that a combined system using a clustering step and an auto-ml trained predictor was novel enough and possibly effective. I decided that the clustering approach should attempt to discern something about the *shape* of trading in gold and another source of market information (the S&P 500) over the past few days and the predictor use that same data to estimate future prices. The derived features used were intentionally kept to a small number so that clustering could be more effective.

Interestingly, by increasing the number of clusters, the system seemed to perform better. This is despite the worsening RMSE. This suggests that the Tabular

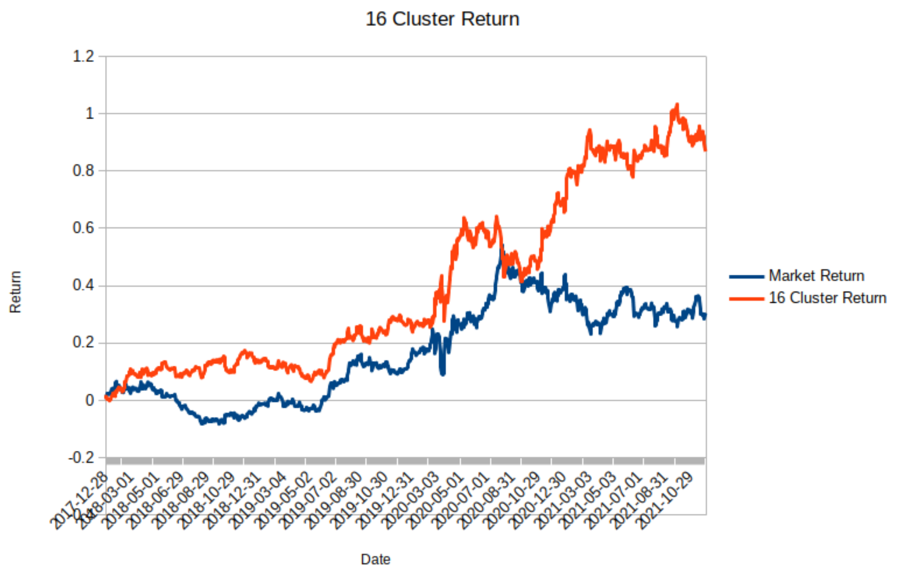

Figure 14: 16 Cluster Return

Predictor was not very effective, but that potentially the clustering was somewhat effective.

The challenges associated with deploying an AutoGluon Tabular Predictor was not insignificant, but ultimately the availability of suitable sample scripts made it somewhat manageable. Another area of challenge was the AWS limit of 30 servers, making it impossible to train and deploy a 32 cluster system. It is possible that such a system would give better results.

Ultimately, the final result was in keeping with my expectations. It was highly unlikely that I would find a model capable of beating the market significantly, and while I suspect that the EMH is not completely correct, the likelihood of finding an exploitable trading strategy is rather small. I achieved results essentially in line with my expectations.

Improvement

There are a number of improvements I feel could be made to this system:

- Add other sources of market information, not just one (S&P 500). These could include other commodities such as oil, silver, etc. or other market indices, such as the Dow Jones Industrial, FTSE 100, DAX, Nikkei 225, ISEQ, etc.
- Add sources of non-market information, such as public sentiment, references to gold in media headlines, etc.

- Use far more of the historic market data for the Tabular Predictors, not just the same derived features used for clustering, but the actual prices going back many days. The limiting of the derived features as used for the clustering makes sense to keep the degrees to a reasonable level and create useful clusters, but that same strategy does not apply for the Tabular Predictors which would likely have benefitted from having more features.
- Train the system on the data to the date for which the prediction is being made, ultimately not using the traditional training/test split. This would allow the system to reflect the latest information while not having any future peeking taking place.
- Train and test system to predict other commodities and market prices.

Ultimately, there are many avenues for possible improvements, and I do think that the approach of using K-Means clustering followed by another predictor is promising, not just in the area of predicting market prices, but to provide systems trained to handle other tasks where the input may be even more readily divided into clusters, and having predictors trained to be specialists in their own cluster would be more useful. For instance, a system designed to identify vehicles could use clusters that ultimately correspond to the different vehicle classifications, and another predictor which is an expert in identifying within that particular class of vehicle.

References

- Gold's history as a currency standard. (2010). Retrieved from https://www.reuters.com/article/idINIndia-52748720101108
- Fama, E. F. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. The Journal of Finance, 25(2), 383–417. https://doi.org/10.2307/2325486
- Chen, J. (2021). Safe Haven. Retrieved from https://www.investopedia.com/terms/s/safe-haven.asp
- Raghuram, K. S. Statistical, Machine Learning predictive analytics and the impact of stock market indicators in predicting gold prices.
- Riazuddin, M. Predicting Gold Prices Using Machine Learning (2020). Retrieved from https://towardsdatascience.com/machine-learning-to-predict-gold-price-returns-4bdb0506b132
- Shah & Pachanekar (2021). Gold Price Prediction Using Machine Learning In Python. Retrieved from https://blog.quantinsti.com/gold-price-prediction-using-machine-learning-python/
- Datta P. (2021). Building A Gold Price Prediction Model Using Machine Learning. Retrieved from https://www.analyticsvidhya.com/blog/2021/07/building-a-gold-price-prediction-model-using-machine-learning/
- Cloud Training with AWS SageMaker. (n.d.). Retrieved from https://auto.gluon.ai/dev/tutorials/cloud_fit_deploy/cloud-aws-sagemaker-training.html
- Deploying AutoGluon Models with AWS SageMaker. (n.d.). Retrieved from https://auto.gluon.ai/dev/tutorials/cloud_fit_deploy/cloud-aws-sagemaker-deployment.html