Price-Pure Prediction of Daily Price Changes in Binary Event Contracts

Stanford CS 229 Project

Adrian Molofsky
Department of Computer Science
Stanford University
molofsky@stanford.edu

Steve Mendeleev
Department of Math and Computer Science
Stanford University
steveroy@stanford.edu

Abstract

Binary event contracts are a popular, unique asset whose prices are emotion-fueled & reflect perceived likelihoods of events. In this study, we distill down to price-pure data & explore how well various models predict daily prices, concluding CatBoost is best.

Keywords: Mathematical Finance, Linear Regression, Gradient Boosting, Risk Management

1 Introduction

Prediction markets (like Kalshi and Polymarket) have become wildly popular recently and historically show a remarkable efficiency in reflecting the perceived likelihood of real-world events, in theory and practice (Wolfers and Zitzewitz, 2006). Accordingly, the future derivatives they offer the public to trade on, event contracts which have binary "Yes"/"No" bet options and resolutions, have become a popular and active asset-class that brings its own set of inherent financial behaviour.

Concretely, as a new control result for event contracts, we aim to explore whether the innately sentiment- and news-driven prices of event contracts can be accurately predicted with machine learning techniques *purely* on the basis of time-price data, and which models execute better. Indeed, we make an intentional choice to restrict ourselves only to price (and date) data, so as to set a benchmark for future work seeking to improve performance that incorporates additional non-price data (like sentiment, news or volume).

Explicitly, our algorithm takes as input the historical time-series of the daily closing prices of a particular event contract as a CSV. We then use linear regression, a decision tree, and various gradient-boosting models to output a predicted percentage price change of the next day's closing price with respect to today's. We determine which kinds of model architectures perform well after tuning.

2 Related Work & Preliminary Experiment

Due to how recently modern event contracts became mainstream, it is difficult to find papers that attempt a similar task with machine learning. However, similar techniques in the *stock market* have frequently been tried, and the area is adequately similar, so we examine related work there.

2.1 One-day ahead movement prediction using beyond-price data

One similar work was where Weng et al. (2017) trained on predicting one-day ahead closing price changes for stocks. They acknowledge and leverage multiple types of data, combining and distinguishing for example from price-pure data and external data like technical indicators and Google News. However, the paper solely trains and tests on a single stock (AAPL) so their model does not have breadth and is likely overfitting to the unique tendencies of Apple stock, reducing its general utility.

Even more crucially, we note that Weng et al. exclusively looked at the *binary classification* task of predicting price movement (i.e. whether price strictly increased or not) which reduces complexity significantly. Although the collective usage of different nature sources is clever and it is encouraging that they achieve percentage accuracy metrics hovering around the 80s with models we intend to use (Weng et al., 2017), this is a far simpler task than price prediction. Indeed, as a preliminary baseline, we ourselves conducted our own classification experiment, training logistic regression on a dataset of various Kalshi event contracts and achieved post-tuning $\sim 84\%$ accuracy. Albeit our dataset was small and hourly-discretized (during which event contracts are arguably more monotonous), the paper's reported

accuracy seems not unachievable with price-pure data sources.

2.2 Comparison of ARIMA and ANN Models for Price Prediction using Price-Pure Data

Alternatively, Adebiyi et al. (2014) also predicts daily stock price changes yet trains on various stocks and speciifcally predicts price *change* also purely based on price data/ One useful decision made by Adebiyi et al., differing from our work, is the decision to include intra-day price features which give a more holistic picture; as for us, the time granularity of price data from our source varies by event contract so for consistency we opted for daily price data.

However, the limitations of the study are twofold: that only two models, an ANN and ARIMA, are tested just to replicate and validate existing results, without additional observations about other models; and testing is only done on a single particular (DELL) stock index (Adebiyi et al., 2014). Nonetheless, having a third-party baseline forecast error for us to benchmark ourselves against is valuable.

2.3 Similar Task for Futures

More related to event contracts than stocks are futures. A paper by Zhao (2021) focuses on predicting price changes in agricultural futures. Granted, the paper involves comprehensive feature engineering and focuses on comparing performance between different model architectures. Yet the study once again suffers from the same limitations as before: technical indicators like moving average are used and the focus is exclusively on evaluating performance within ARIMA models, giving little breadth. Still, the approach is refreshing and clever in its usage of PCA.

Similarly, Waldow et al. (2021) focus instead on the binary classification task for futures with similar drawbacks, though have interesting ideas of rolling window and backtesting approaches.

3 Dataset and Features

3.1 Data Collection

Because of Polymarket and Kalshi APIs not offering historic pricing data, we manually downloaded the CSV files of several both active and already-resolved binary prediction markets on Kalshi, which purely contain all-time prices ($0-1) and the respective timestamps. As mentioned earlier, the structure of said data files varied drastically between event contracts. To stay consistent we stuck with daily data (taking the last recorded price on any given day to be the closing price).

3.2 Data Selection

We wanted to reduce bias in event contract data selection as much as possible to ensure our final model(s) would be as generalizable as possible and not overfit to a particular topic, time-range, price-range and more. To accomplish this, we manually went through each of Kalshi's overarching market themes, filtered contracts by highest volume, and took the first several event contracts or so. Our decision to specifically opt for higher volume event contracts was to avoid the common syndrome of unpopular event contracts where price can remain constant for extended periods of time, simply due to lack of active market participants. Intuitively and implicitly, we want our model to assume it is predicting prices for event contracts that are driven by actual trader involvement, and since our model aims to be market-question-agnostic it wouldn't otherwise be able to infer lack of market participants from price data alone.

3.3 Pre-processing

It is worth mentioning some of the pre-processing that we conducted. Firstly, the last row of each individual event contract's CSV was removed as it, by default, was set to the (possibly older) datapoint that the cursor was last one. Secondly, for event contracts with several sub-events (e.g. "Which video game will win Best Game 2024 ?" has multiple sub-events for the > 2 games in the running), we would always pick to keep the time-series data of the first sub-event listed in the CSV as an approximately random selection process (otherwise including multiple of them could mess with the independence of data as two sub-events of the same broader contract are necessarily correlated).

3.4 Data Pipeline

We created a Python pipeline to traverse all downloaded CSV files and construct a single principal dataframe with the market question, date (day), closing price, and all additional features we engineered as described below; additionally, we evidently included the target output which is the one-day ahead fractional change in price, defined for the ith day as $\frac{\text{close}(i+1)-\text{close}(i)}{\text{close}(i)}$. Altogether we extracted and preprocessed 66 distinct CSVs to create a final CSV with $10,065$ total example data points, with 20 features.

3.5 Feature Engineering

As we sought a price-pure approach to set a first baseline for modern event contracts in general, we had to engineer all of our features from dates and prices alone. To that end, we ultimately came up with 20 features that appeared correlated and valuable to the task at hand. Features were chosen for their ease of extraction from the limited time-series data we already had as well as ensuring they intuitively could have a bearing on pricing, such as whether we're in a weekday, rolling mean over the last week, and the volatility ratio. Here's a sample of the final dataset with a few of the feature columns:

Market Question	price	mean_7d	date	target
U.S. debt default < 2024?	.146	.128	2023-03-14	.0521
GPT = Best AI 2024?	.660	.677	2024-11-26	-.0186
Costco hotdog Δ price < 2026?	.22	.213	2024-06-27	.227

Figure 1: Sample of processed, final dataset. Includes 3 distinct event contracts with 3 of the 20 features and the sole target output: current price, 1 week rolling mean, date and signed fractional price change.

4 Methods

4.1 Linear Regression

Linear regression attempts to model a linear relationship wherein it tries to learn weights $\theta = (\theta_1, \ldots, \theta_d)$ for d features such that the hypothesis function, evaluated on an input $x = (x_1 = 1, \ldots, x_d)$ (we take $x_1 = 1$ so that we can have a non-origin intercept term), $h_\theta(x) = \theta_1 x_1 + \cdots + \theta_d x_d = \theta^T x$ is as "close" on average to actual values as it can be. Concretely, this "closeness" is induced by optimizing the mean square error, which we take to be our objective function, or simply the average square of how far off our hypothesis prediction is from the actual output over all n examples; that is,

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$$

where $x^{(i)} \in \mathbb{R}^d$ is the ith example vector and $y^{(i)} \in \mathbb{R}$ is the actual output observed for that input.

One can certainly optimize the objective function in the traditional fashion that most other machine learning algorithms do, by selecting a learning rate and starting point then repeatedly applying gradient descent (and we benefit from this objective function being convex). However, linear regression is a particularly neat algorithm in that it has a closed-form solution that one can derive in matrix form, called the normal equation, $\theta = (X^T X)^{-1} X^T \vec{y}$ where each row in $X \in \mathbb{R}^{n \times d}$ is an example $x^{(i)} \in \mathbb{R}^d$ and $\vec{y} \in \mathbb{R}^n$ is the list of actual outputs.

4.2 Single Decision Tree

A single decision tree is a non-linear predictive modeling approach that partitions the feature space through recursive binary splits, effectively distilling both discrete and continuous predictions down to answering finitely many binary questions. Unlike linear regression's single hyperplane, a decision tree creates a tree-like model of decisions where internal nodes represent feature tests (e.g. is feature $x_1 > 5$?), branches represent test outcomes, and leaf nodes represent predictions or class labels.

The objective function to be optimized for a single regression tree minimizes the variance within splits, or

$$J = \sum_{m=1}^{M} \sum_{i \in R_m} (y_i - \hat{y}_m)^2$$

where M is the number of terminal nodes; R_m is the set of instances in the m-th terminal node, and $\hat{y}_m$ is the mean value for instances in node m. Intuitively but also algorithmically, tree construction is implemented to follow a greedy, top-down approach wherein we repeatedly select the best feature and split point that most reduces variance;p recursively apply the splitting to each generated subset; and ultimately stop when we have reached a pre-defined stopping crtierion.

With respect to linear regression, a single decision tree is highly versatile as it can be used for both classification and regression, has no inherent assumption about the shape of the date (e.g. linearity), is a lot more interpretable in its outputs yet often tends to overfit.

4.3 Gradient Boosting for Regression

Gradient boosting for regression is a form of ensemble learning that relies on multiple weak learners, typically decision trees, to produce stronger predictions. The algorithm is highly effective at recognizing complex patterns in data and can model non-linear relationships, making it well suited for predicting asset prices. The algorithm minimizes least squares loss by sequentially adding predictions for the weak learners to update the gradient. Mathematically, the update for the model is defined as $F_m(x) = F_{m-1} + \eta \cdot h_m(x)$

where $F_m(x)$ is the model update after the m-th iteration, F_{m-1} is the previous model, η is the

learning rate, and $h_m(x)$ is the weak learner. The least squares loss function is defined as

$$L(y, F(x)) = \frac{1}{2}(y - F(x))^2$$

which quantifies the difference between the predicted and target values. At each step, the algorithm computes the residuals for the weak learner $r_i^{(m)} = y_i - F_{m-1}(x_i)$ for all of the data points and trains a weak learner using the residuals. It then updates the model. The sequential nature of the algorithm ensures that each new weak learner focuses on a feature of the dataset which the previous model did not, improving the predictive capability and reducing the potential for underfitting. Optimizations of the algorithm include:

4.3.1 Histogram-Based Gradient Boosting

Histogram-Based Gradient Boosting improves on traditional Gradient Boosting by relying on bins to optimize the training time for each of the decision trees. The algorithm stores continuous feature values in discrete bins that allows the method to operate on large-scale applications with significantly reduced memory over head. Since it simplifies the process of splitting decision trees by reducing the decision possibilities to only bin values, this method is well suited for large datasets and dealing with applications in which memory is limited and efficiency is a high priority.

4.3.2 Categorical Boosting

Categorical Boosting is an advanced boosting method that utilizes a numerical representation for categorical variables. The process converts categorical features into an encoding in a specified order, allowing features to be easily differentiated from one another when making predictions. The predetermined ordering of the method allows it to generalize well to unseen data points, reducing the potential for overfitting and significantly improving the performance over other boosting methods.

5 Experiments & Results

5.1 Metrics

Our primary metrics are mean square error (MSE), mean absolute error (MAE) and mean forecast error (MAE). The latter is a useful metric we borrow from Adebiyi et al. (2014) and will use to explore the best model. It is the mean average over all test examples of the forecast error $\text{FE} = \frac{\text{actual} - \text{predicted}}{\text{actual}}$.

5.2 Linear Regression

After training a vanilla linear regression model with the standard sum of squared errors objective function and derived normal equation, we obtained an MSE of 0.0446. Thereafter, we repeatedly pruned the least correlated features which we deemed to be those that had magnitude of coefficient < 0.1. The rationale for why certain features had such insignificant correlation could very well be that they were redundant and what information they possessed was already communicated through other similar or proxy features, or simply that they statistically had little to no bearing. We observe below how the feature importance coefficients changed are all sufficiently relevant:

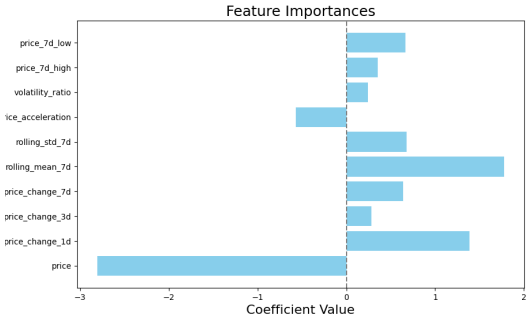

Figure 2: Feature importance $[-\infty, +\infty]$ by feature, post-pruning.

This alone reduced the MSE to 0.0425 and it already seemed feature-wise that we were plateauing in reducing error. Evidently, there is underfitting going on because a linear model is not suitable to predicting this natured financial data, and we are witnessing high bias. There is not much more we can do to squeeze out additional reduced error, since the typically most simple approach to doing so is to add an L2 regularization term; this does not apply here as we are nowhere near overfitting and, much to the contrary, underfitting. The only way forward from underfitting, provided we are restricted to the features and dataset we have, is increased model complexity.

5.3 Single Decision Tree

For the single decision tree, we opted for k-fold cross validation and specifically a value of $k = 10$ as this is what was cited as very efficient in several temporal stock price prediction papers (Weng et al., 2017), yielding a good balance between performance and time taken. After additionally pruning several features that were not significantly correlated with the output, the model only achieved a

rather poor MSE of 0.305, which is a lot weaker than the linear regression baseline.

After collecting both training and testing metrics, we observed the testing MSE was far higher than that of training error which was relatively low, signalling at overfitting which we already know decision trees are prone to doing.

To reduce overfitting, we implemented a grid search, which is a hyperparameter fine-tuning technique, for a handful of parameters which exhaustively constructs, trains and tests all possible decision trees with distinct combinations of parameters and reports which does best. The optimal model resulting from grid search was still only performing at MSE of 0.147, hinting that more complexity is needed, such as ensemble learning that combines many decision trees.

```
param_grid = {
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5],
    'max_features': ['auto', 'sqrt'],
    'criterion': ['squared_error']
}
```

5.4 Gradient Boosting for Regression

The method was well suited for modeling asset price swings since it could robustly identify non-linear patterns in financial data and the combination of weak learners recognized complex patterns, outperforming less sophisticated models. Because of their ability to reliably identified trends and complex patterns within contracts, these were the best preforming models. The best models were found using randomized search cross validation and split the dataset into 5 folds, one for testing and the others for training to discover the optimal parameters. Gradient Boosting for Regression used the default squared error as a loss function, a learning rate of 0.001, 100 estimators, and a max depth of 7. These parameters were found via randomized search.

5.4.1 Histogram-based Gradient Boosting Regression

Histogram-based Gradient Boosting used squared error as a loss function, had 15 as the number of max leaf nodes, relied on l2 regularization with a value of 0.9385527090157502, had a learning rate of 0.0010778765841014328, and ran for 376 iterations. This method ran for a shorter time than any of the other gradient boosting algorithms, due to its binning optimization.

5.4.2 Categorical Boosting

The model used root mean square error as a loss function, had a max depth of 4, learning rate of 0.00171, and ran for 121 iterations.

6 Evaluation & Discussion

Model	MSE	MAE
Lin Reg	0.0425	0.0979
Dec Tree	0.147	0.0918
GradBoost	0.0397	0.0840
HistoBoost	0.0395	0.0845
CatBoost	0.0395	0.0835

Figure 3: Final testing results by model architecture.

We see based on MSE and MAE that CatBoost was best. The ordered boosting approach allowed the model to generalize well to unseen data and effectively learn features which might otherwise be overlooked. In addtion, ordered boosting greatly reduced the ability of the model to overfit the training dataset, and to further address overfitting, the designers of CatBoost permute the dataset to produce more robust predictions. Using a maximum depth of 4 for the training of CatBoost also prevented overfitting.

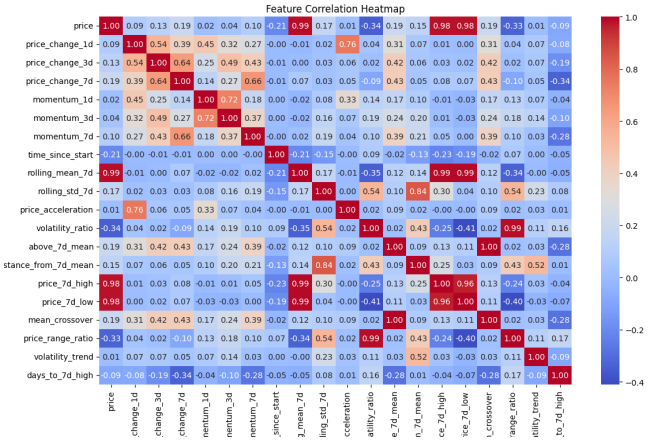

Figure 4: Covariance matrix for CatBoost.

7 Conclusion / Future Work

We trained various models to compare event contract pricing, seeing CatBoost is best. The rationale is likely a single decision tree is not complex enough and overfits, whereas ensemble learning rounds it out. In the future, we would like to go beyond price-pure data and go deeper and more broad in model architecture. It seems the real value lies in more non-price data and less interpretable models.

8 Contributions

Steve worked a lot on the data collection, pre-processing and feature-engineering pipeline while Adrian focused on creating an evaluation pipeline for trained models. Steve conducted preliminary binary classification experimentation with logistic regression prior to the switch to the more granular percentage price change target.

There, Steve trained and tuned the linear regression model and unit decision tree, while Adrian did so for the gradient-boosting models. Together, both Adrian and Steve generated plots for the dataset and computed evaluation metrics to measure the performance. Both team members contributed actively to the shared GitHub repository where all scripts and data live.

References

Ayodele Ariyo Adebiyi, Aderemi Oluyinka Adewumi, and Charles Korede Ayo. 2014. Comparison of arima and artificial neural networks models for stock price prediction. *Journal of Applied Mathematics*, 2014(1):614342.

Fabian Waldow, Matthias Schnaubelt, Christopher Krauss, and Thomas Günter Fischer. 2021. Machine learning in futures markets. *Journal of risk and financial management*, 14(3):119.

Bin Weng, Mohamed A Ahmed, and Fadel M Megahed. 2017. Stock market one-day ahead movement prediction using disparate data sources. *Expert Systems with Applications*, 79:153–163.

Justin Wolfers and Eric Zitzewitz. 2006. Prediction markets in theory and practice. national bureau of economic research Cambridge, Mass., USA.

Hailei Zhao. 2021. Futures price prediction of agricultural products based on machine learning. *Neural Computing and Applications*, 33(3):837–850.