Approximating the Decision-Making Process of Buy-Side Bond Traders

Sean Fuller
Trader/Lead Quantitative Developer, Invesco
Northwestern University
August 2023
Sean.Fuller@Invesco.com

1 Abstract

In the evolving landscape of financial markets, the bond market — the world's largest non-currency market — lags in its adoption of automated trade execution strategies, particularly on the buy-side. The multifaceted and illiquid nature of the bond market presents challenges to the implementation of traditional analytical methods. This study examines the intricate decision-making process of buy-side bond traders and suggests that machine learning (ML) models may offer a more effective representation than expert systems and heuristic rules. Utilizing a real-world dataset collected from a buy-side fixed income trading desk over the course of a year, various ML models are trained and compared in a demonstration of both the potential and the challenges of embedding ML within a live trading environment. Beyond model efficacy, the study emphasizes the critical importance of scalability, robustness, and the preservation of high execution standards in real-world software systems, with the chosen model being actively utilized in a production trading environment. Covering the entire machine learning pipeline from data acquisition to deployment, this paper aims to be accessible to and useful for buy-side bond market participants, data scientists, and ML engineers alike.

2 Keywords

Bond Trading, Financial Machine Learning, Automated Execution, Gradient Boosted Decision Trees

3 Introduction

The bond market, despite its vast scale, has historically lagged behind its stock market counterpart in terms of embracing automation in trading. The sell-side, equipped with sophisticated pricing and market-making algorithms, contrasts with a buy-side that remains gradual in its adoption of automated strategies. This hesitancy is not without reason; the bond market's fragmented and illiquid nature makes the application of traditional analytical methods challenging. Nevertheless, the ability of a buy-side bond trader to articulate the reasoning behind every trade hints at an underlying decision function. This function, however, seems to elude simple heuristic representations and expert systems, suggesting the presence of complex decision boundaries within a high-dimensional feature space. This complexity signals an opportunity for ML approaches to step in and capture these elusive decision-making intricacies.

While machine learning models have been lauded for their ability to model non-linear and complex relationships, a closer look reveals their kinship to classical models such as multiple linear regression, albeit with certain nuances. This paper seeks to formalize the representation of the buy-

side bond trader's decision-making process. Through rigorous comparisons, we select an optimal model trained on a dataset curated from a tumultuous year marked by significant shifts in interest rates, ensuring the model's robustness. Beyond just identifying the right model, we also tackle the real-world challenges of integrating it into an active trading environment.

The central aim of this study is to present a comprehensive case study on the potential integration of ML methods into the buy-side bond trading landscape. When designing the proposed hybrid automated execution ("auto-ex") workflow, we emphasize a scientific approach that upholds transparency, thoughtful and explicit decision-making around tradeoffs, and business scalability without compromising on execution standards. The paper delves deeply into the practicalities of deploying the chosen model, detailing the challenges and solutions associated with integrating it into a live trading environment. The system's practical relevance and efficacy are underscored by its substantial utilization in production (being used to review 5,500 trades in the first month of deployment), with conservative projections indicating it will execute greater than $3.6 billion in trading volume between its launch in July 2023 and July 2024.

4 Literature Review

The burgeoning field of financial machine learning boasts a wealth of research, but a significant portion of this research is heavily oriented towards portfolio optimization. While trade execution is another area of focus within the literature, its treatment differs considerably from the problem we examine in this paper. Traditionally, trade execution is framed as a control problem. The core objective is to discern optimal policies within dynamic systems, particularly equity market limit order books. Such formulations seek to achieve complete and best execution over a specified time period while minimizing the market impact of one's own trades on the price of the traded security, highlighting the feedback mechanism inherent in such environments. Capturing this feedback dynamic, especially within the multi-time step element typical of control problems and reinforcement learning (RL) formulations, is crucial for modeling trade execution in liquid markets like stocks.

However, the bond market, the focus of our study, operates on a different paradigm. Unlike stock exchanges, bond trades are typically finalized through a reverse auction or request-for-quote format. In this setup, the influence of past actions is generally negligible since trades are typically completed all-or-none (AON), removing the complex feedback loops in stock trading that necessitate RL approaches. This distinction underscores the unique challenges and opportunities inherent to bond trading, which sets it apart from the more commonly studied stock trading scenarios.

Given this context, our literature review aims to familiarize financial industry practitioners (particularly those well-versed in linear modeling) with the classification models employed in our subsequent model selection process in Section 7. We emphasize models that not only offer predictive accuracy but also provide insights that resonate with industry practitioners. Given the crucial role of interpretability in the financial domain, models are presented in decreasing order of inference explainability. Predictions of the later models are more accurate but less interpretable, and we highlight the trade-offs between accuracy and interpretability.

4.1 Motivation

Multiple Linear regression is a model formulation with which traders and asset management professionals are deeply familiar. It forms the basis of the Chartered Financial Analyst (CFA) curriculum and is the dominant framework used in investment industry research. All methods discussed in the remainder of the paper can be viewed as an adaptation of linear regression and

will be presented with frequent reference to that analogy.

The decision-making process that this paper strives to model is whether or not to execute a trade based on a set of numeric feature values that describe the trade (trade size, direction, price of the best level, etc.). This can be modeled as a binary classification problem ($K = 2$), where the target variable is a binary decision variable with EXECUTE as the positive class ($k = 1$) and PASS as the negative class ($k = 0$). Given n observations of p features, the dataset can be expressed as $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, where $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T$ represents the vector of numeric features for trade i and y_i is a scalar indicating the actual decision taken by the trader.

In this context, the following models can then be thought of as methods for approximating $\hat{f}$ to achieve a high classification accuracy. Specifically, $\hat{f}$ in the binary classification setting is generally expressed as the conditional class probability:

$$\hat{f}(X) = \hat{\Pr}(Y = k \mid X = X_0) \doteq \hat{p}(X) \qquad \text{(Eq. 1)}$$

A threshold value is then chosen to make the model select a class membership for each observation. For example, if the model predicts
$\hat{p}(X_0) = \hat{\Pr}(Y_0 = \text{EXECUTE} \mid X_0 = \{\text{ trade\_size }_0, \text{ best\_price }_0\}) = 0.6$, the final prediction would be EXECUTE at a threshold of 0.5 and PASS if the threshold had instead been set at 0.8. In short, the following models predict the probability of class membership ($\hat{f}(X_0) = \hat{p}(X_0)$), whereas linear regression models directly predict a numeric target ($\hat{f}(X_0) = \hat{y_0}$). The chosen models represent the most common choices for modeling binary classification and are presented in descending order of inference interpretability.

4.2 Highly Interpretable Models

4.2.1 Logistic Regression

Logistic regression is a direct variation on linear regression that enables the prediction of class membership probability rather than of a numeric value. Attempting to model a binary target using linear regression can result in probability estimates that fall outside of the range $[0, 1]$. To address this, logistic regression replaces the linear formulation with the S-shaped logistic (sigmoid) function, which *is* bounded by $[0, 1]$:

$$\hat{f}(X) = \hat{p}(X) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 X_1 + \cdots + \hat{\beta}_p X_p}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 X_1 + \cdots + \hat{\beta}_p X_p}} \qquad \text{(Eq. 2)}$$

The right-hand side of this equation can be made linear by converting the logistic function from probability form to odds form and taking the natural logarithm:

$$\log\left(\frac{\hat{p}(X)}{1 - \hat{p}(X)}\right) = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \cdots + \hat{\beta}_p X_p \qquad \text{(Eq. 3)}$$

While the resulting equation (called the log-odds or logit) looks similar to linear regression, the difference in the left-hand side gives rise to some important differences in interpretation of the parameter values. In the linear case, the ceteris paribus effect of X_j on Y is captured directly by $\hat{\beta}_j$ (assuming a p-value $< \alpha$). In Equation 3, however, $\hat{\beta}_j$ describes not the change in $\hat{p}(X)$ associated with a one-unit change in X_j, but rather the associated change in the log-odds ratio. The interpretation of this is that in logistic regression, the effect of X_j on $p(X)$ varies based on the value of X. The signs of the parameter values in Equation 3 still convey directional information about the relationship between X_j and $p(X)$.

The logistic function has roots dating back to models of population growth developed in the early 1800s, but modern techniques of logistic regression are typically attributed to Cox (1958). Unlike the other models presented, logistic regression is a parametric method, meaning that the number of parameters remains fixed regardless of the amount of data being modeled. It assumes that the data-generating process exhibits a relationship between the features and target that conforms to the functional form expressed in Equation 2. This simplifying assumption results in a higher degree of interpretability at the cost of lower prediction accuracy if the true relationship being modeled deviates from the assumed form or displays a complex hierarchical structure.

4.2.2 Decision Trees

First introduced by Quinlan (1986), decision trees are interpretable non-parametric models capable of modeling complex phenomena. By repeatedly splitting the dataset based on feature partition thresholds that maximize class purity in the resulting segments, decision trees partition high-dimensional datasets in a manner that captures hierarchical feature interactions. Once defined, the class membership of unseen observations can be predicted by following the series of if-then statements that define the tree's structure.

Decision trees are commonly fit using the CART algorithm (Breiman et al. 1986) that recursively performs a greedy search across features for the split that produces the minimum class impurity in the resulting regions. The algorithm measures impurity resulting from a split using either entropy or Gini Impurity:

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2 \qquad \text{(Eq. 4)}$$

This is repeated until no split can reduce impurity or a predefined stopping criterion is reached. The resulting structure consists of a root node (where the first split is made), child nodes, and leaf nodes. The root and child nodes each have a decision rule t that splits on a single feature k. Once fit,$\hat{f}(X)$ can be expressed as:

$$DT(x) = \begin{cases} \text{Value}_{\text{ leaf}} & \text{if is leaf node} \\ DT_{\text{left}}\,(x) & \text{if } x_i \leq t \\ DT_{\text{right}}\,(x) & \text{otherwise} \end{cases} \qquad \text{(Eq. 5)}$$

Conceptually, the algorithm can be thought of as repeatedly splitting the feature space based on class membership of observed data to reduce entropy/maximize information gain at each step. This is an algorithmic embodiment of a powerful concept that aligns well with real-world implementation, as will be discussed in Section 8.2. The flexibility of this approach in modeling complex phenomena is also its biggest downside if not properly controlled. Because decision trees infer the form of $\hat{f}$ from the data rather than enforcing a pre-specified functional form, the complexity of trained models (as measured by depth of the tree) grows with the size and complexity of the training data. In the absence of safeguards, the model will encode decision rules arising from noise in the dataset (such as a fat-finger or improperly calculated analytics), leading to poor performance on unseen data.

4.2.3 K-Nearest Neighbors

The k-Nearest Neighbors (KNN) algorithm (Cover and Hart 1967) is a non-parametric, instance-based learning algorithm that simply classifies a new observation according to the most

common class of the closest K data points in the training dataset (the "neighborhood," denoted N_0):

$$\hat{p}(X) = \widehat{\Pr}\left(Y = j \mid X = x_0\right) = \frac{1}{K} \sum_{i \in N_0} I\left(y_i = j\right) \qquad \text{(Eq. 6)}$$

The choice of K determines the flexibility of the resulting model. Too small a value results in a highly intricate decision boundary that risks capturing noise associated with outliers, while too large a value might smoothen the decision boundary excessively.

Despite being the simplest method considered, the prediction interpretability decreases in practice as the number of features increases. Since the identification of neighbors relies on a distance metric, datasets with dimensionality > 4 naturally lead to challenges in visualizing and understanding these distances.

4.3 Ensemble Methods with Moderate Interpretability

Ensemble methods, as the name suggests, combine multiple models (often termed "weak learners") to craft a single predictive model that's more robust and accurate than its individual components. Such techniques have become increasingly popular due to their ability to reduce the potential overfitting of individual models and aggregate diverse model perspectives, thereby producing predictions that often outperform any of the individual models used.

4.3.1 Random Forest

The Random Forest (Breiman 2001) is a powerful ensemble learning method that uses a collection of decision trees. Unlike a singular decision tree that might be highly sensitive to noise in its training data, a Random Forest trains multiple trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. In particular, Random Forests employ "bagging" (a portmanteau of "bootstrap aggregating"), where each tree is built based on a sample taken with replacement from the training data. Additionally, when determining where to split the data, Random Forests select the best feature amongst a random subset of features. These strategies introduce diversity among the trees and typically result in a more robust model. Given its reliance on decision trees, the Random Forest maintains some degree of interpretability, particularly with the use of feature importance metrics, which rank the importance of input variables based on their predictive power.

4.3.2 Gradient Boosted Decision Trees

Unlike Random Forests which build independent trees, gradient boosting is a sequential process where each tree is built to correct the errors of the prior tree. Continuing the comparison to linear regression, this approach can be thought of as a tree-based equivalent to an additive sequence of linear regressions where subsequent models after the first have as their dependent variable the error of the cumulative prediction of the preceding models.

XGBoost (Chen and Guestrin 2016), an acronym for "eXtreme Gradient Boosting", is a gradient boosting system for tree ensembles that implements L1 and L2 regularization, preventing the model from becoming overly complex. CatBoost (Prokhorenkova et al. 2017), a portmanteau for "Category Boosting", uses a permutation-driven alternative to the classic boosting process called ordered boosting, offering a further reduction in overfitting by avoiding observation-wise target leakage. Both XGBoost and CatBoost are distinguished by their robust and efficient library implementations, with both libraries natively handling null data, CatBoost supporting automatic

encoding of categorical variables without additional preprocessing, and both libraries featuring optimized GPU support for accelerated training. In the realm of model interpretability, both frameworks offer a suite of tools that facilitate insights into feature importances, model predictions, and the interactions among features, thus aiding practitioners in understanding the driving factors behind their predictions.

4.4 Black-Box Models

4.4.1 Multilayer Perceptron/Deep Feedforward Neural Network

Designed to mimic connections between neurons in the brain, a Multilayer Perceptron (MLP) can be envisioned as a network of interconnected, sequentially stacked linear regressions. Each layer in this network contains a set of nodes (neurons), where each node performs a linear combination (weighted average) of its inputs (which are the output from previous neurons). A non-linear activation function (such as the sigmoid function seen in Logistic Regression, for example) is then applied to the result of this linear combination before it is passed to subsequent layers' neurons. The power of an MLP, and neural networks in general, comes from this addition of non-linearity. Without these activation functions, the network mathematically collapses into a single layer of linear transformations regardless of network depth.

As introduced by Rumelhart, Hinton, and Williams (1986), training an MLP involves adjusting the weights of the linear combinations in each node such that the difference between the predicted outputs and the actual outputs is minimized. While in linear regression this can be achieved efficiently using numerical methods such as Ordinary Least Squares, neural networks require backpropagation and optimization algorithms like stochastic gradient descent.

A key concept tied to MLP is that of distributed representation. Each node in a given layer captures some part of the input information. When these nodes are combined, they represent the information in a distributed manner across the layer. As data flows from one layer to the next, these representations get transformed and recombined, allowing for an intricate and hierarchical extraction of patterns from the input data. While this enables the model to automatically and adaptively learn spatial hierarchies of features, it also introduces challenges in discerning the exact patterns they've learned.

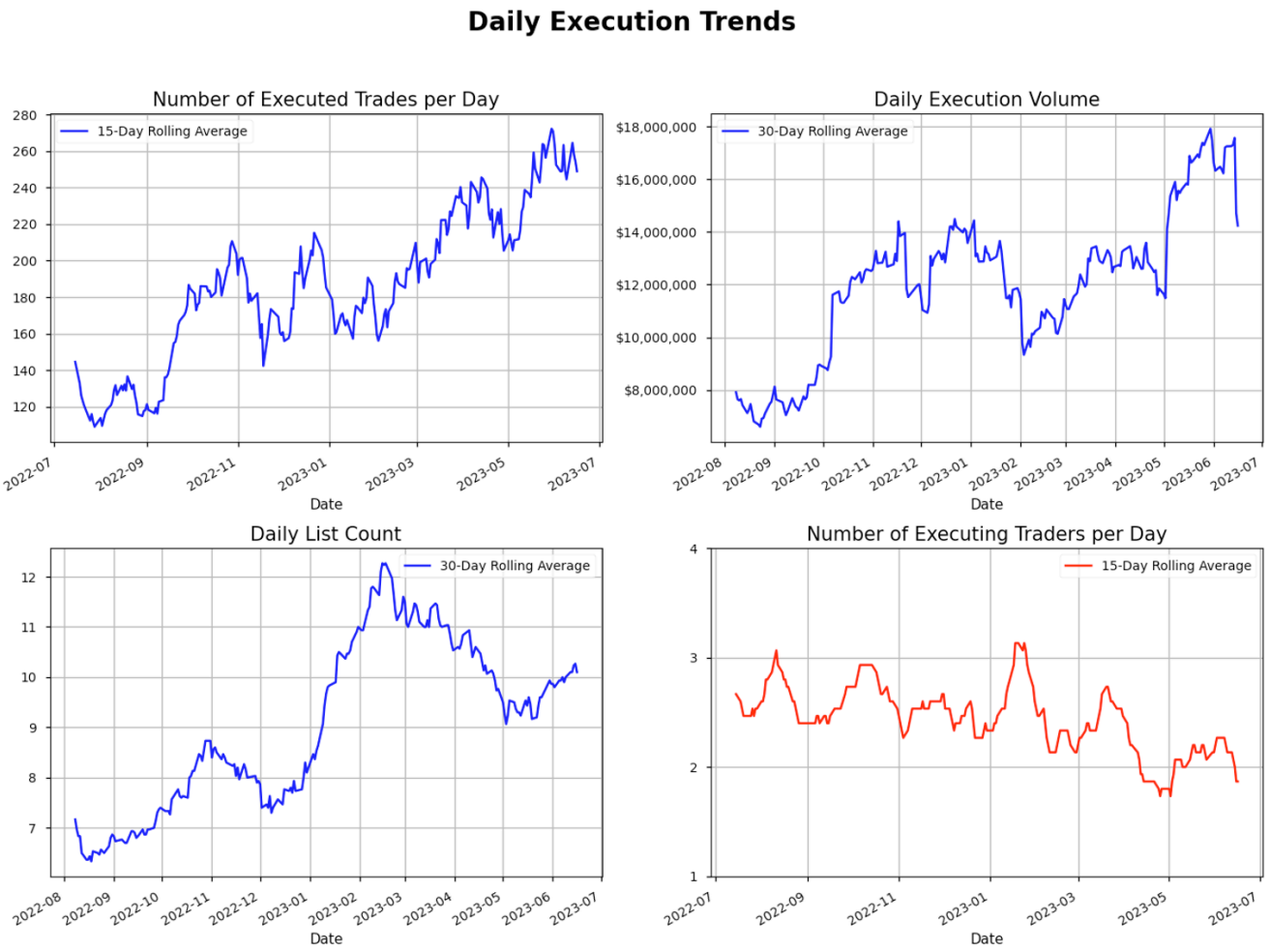
5 Data

5.1 Dataset Description

The dataset used in the experiments is a collection of historical trades. Each observation (row) represents a single trade. The binary target (dependent variable) y represents the decision that was actually taken by the trader (with $y = 1$ indicating EXECUTE and $y = 0$ indicating PASS). Eighteen features are used as predictors of y. These features (independent variables, in the language of regression) include market and item-specific metrics such as intraday CDX change and the number of responses received as well as several standard bond trading analytics such as the best response's price, yield, spread, and their respective differences from the evaluated price (the "eval".)

The dataset was collected over a 12-month period spanning 6/16/2022 to 6/16/2023. These trades represent a subset of the taxable trading activity of Invesco's separately managed account bond trading desk. Trends of the desk's daily execution activity during the observed period are displayed in Figure 1.

Figure 1: Daily Execution Trends 6/16/2022 - 6/16/2023



5.2 Business Context

The trends displayed in Figure 1 demonstrate the motivation behind the model development and programmatic execution efforts described in this paper. In particular, the rising daily trade count and volume juxtaposed with the number of executing traders highlight the need for scalability. This need is further underscored by the fact that the trading workflow captured in this dataset represents only a specific subset of the trading desk's daily responsibilities: electronically executed corporate bond trades associated with the daily inflows and outflows of the approximately 9,000 separately managed accounts (SMAs) managed by the desk.

Beyond this particular workflow, traders are responsible for managing other workflows such as determining and staging the trades to be sent for attempted execution (the desk employs a hybrid trader/portfolio manager approach), "non-electronic" execution (also known as "voice trading," this describes trade negotiation and execution that occurs over the phone or on Instant Bloomberg chat–this remains the predominant approach for larger, block-sized bond trades), trading in other asset classes besides corporate bonds, analyzing markets for relative value opportunities, harvesting tax losses, and monitoring requests from established and prospective clients. In short, embedding model-driven decision-making in the execution process is a necessity for the business to scale to meet the trading activity resulting from increased demand. Moreover, it is highly desirable from the traders' perspective, as the resulting productivity gain frees up time and mental capacity to

focus on more strategic trading and investment management tasks.

5.3 Data Collection and Initial Expert Systems Approach

Prior to the collection period (i.e., prior to an automated approach being developed), traders reviewed trades directly in the MarketAxess UI and manually gathered trade analytics as needed for the decision-making process. This process proved unscalable as the desk saw increased volume, and in 2022, a Python program was written to enable a more efficient and systematic method for trade review.

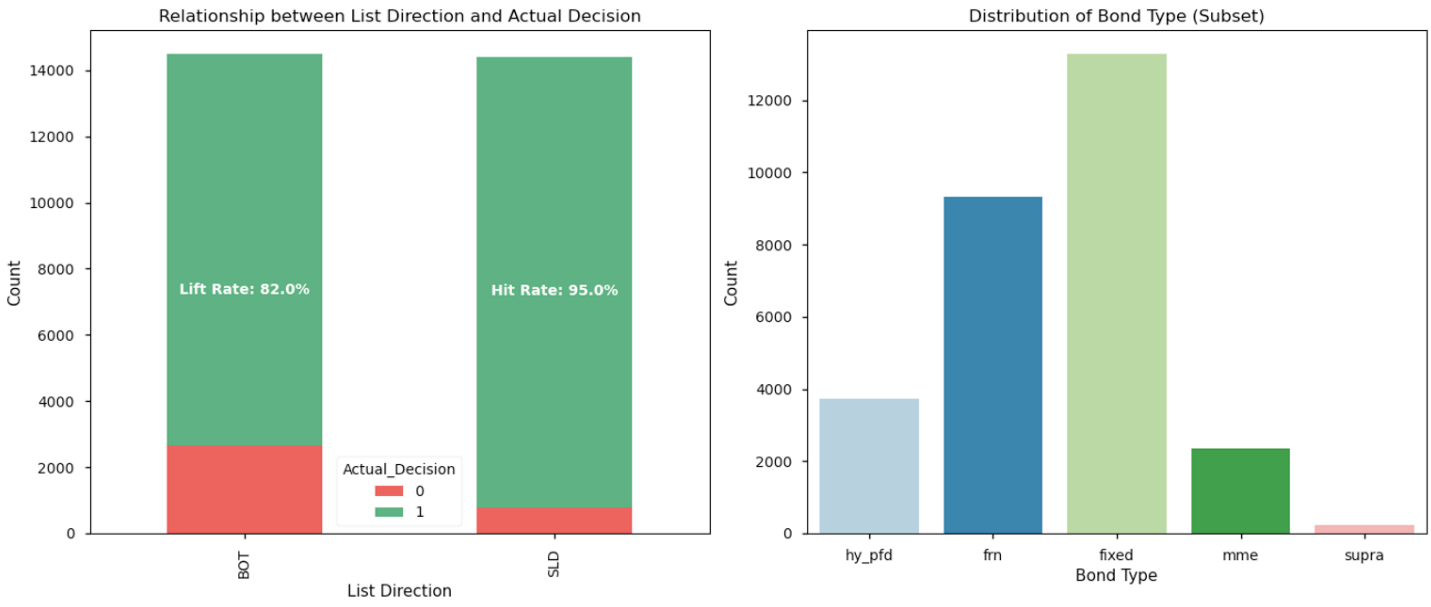
That script ingested a CSV file exported from the MarketAxess UI, collected analytics from several data services via API, and output the result to an Excel spreadsheet. After reviewing the Excel output, traders would execute manually in the MarketAxess workstation. For each trade under review, the program also used a heuristic rule to recommend an action (EXECUTE or PASS). Dissatisfaction with the accuracy of the explicitly programmed classifier (as well as with inefficiencies remaining in the improved workflow as volume continued to increase, as shown in Figure 1) led to the efforts described in this paper.

5.4 Data Cleaning and Exploratory Data Analysis

The output of the described Python program forms the dataset used in the experiments performed here. Although the desk executed 45,543 trades on MarketAxess during the collection period (not inclusive of passed trades), only 35,386 trades were run through the legacy Python program and captured for analysis. Of these, 1,493 were dropped when null values and outliers were removed, and 738 received no response (and thus were removed as no trading decision was made).

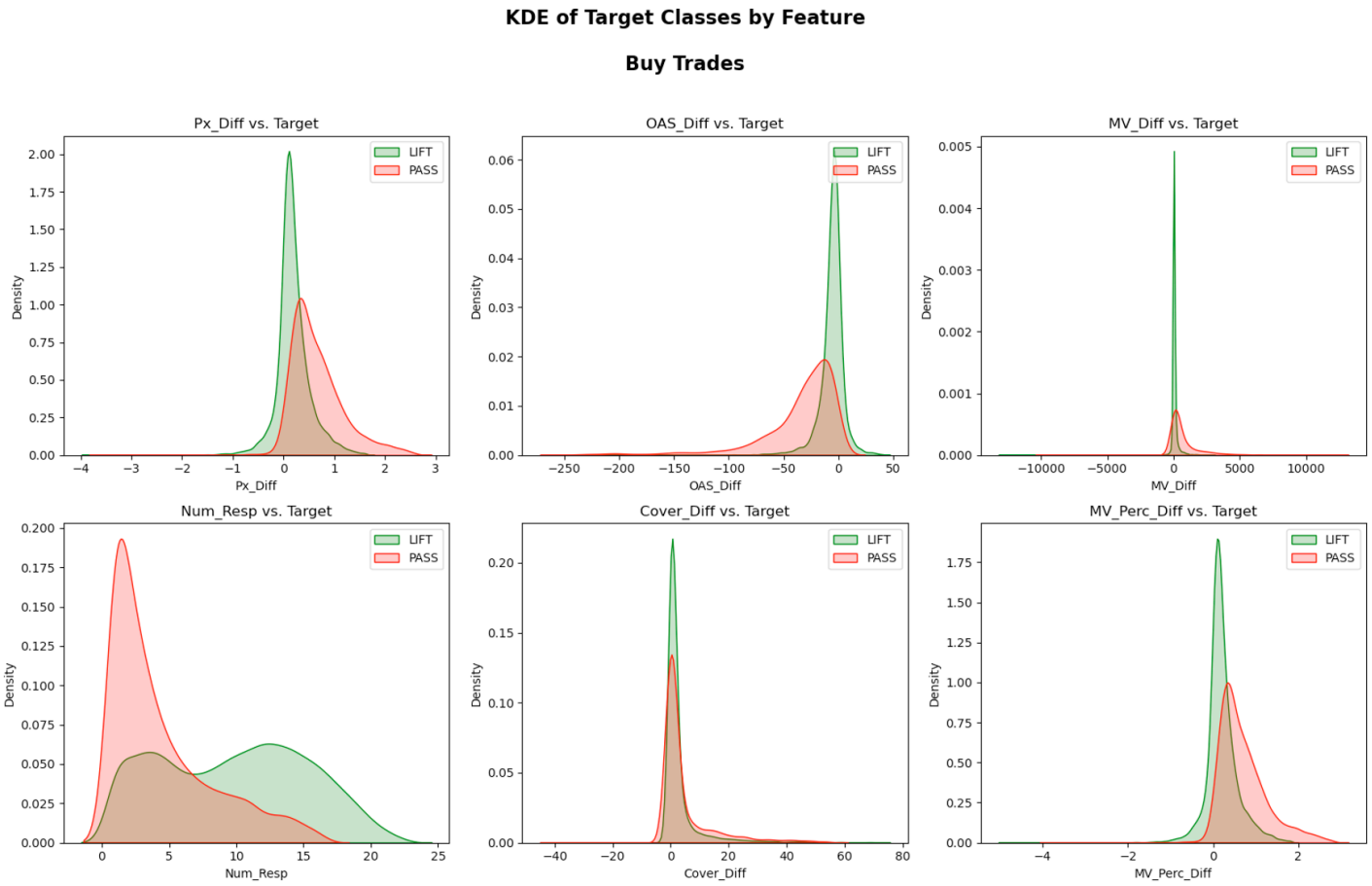
The dataset is expected to be imbalanced towards the positive class, as the majority of trades are executed. However, after reviewing the trades remaining after taking the above cleaning steps, 30% of trades had a target class of PASS (which is higher than would be expected). Upon closer inspection, this resulted from a number of lists with a 100% pass rate (see Section 8.2 for a more detailed discussion). After removing these lists, 29,218 trades totaling $1.65 billion in par value remained, with 88% belonging to the positive class. The class imbalance and bond type frequency of this dataset are shown in Figure 2.

Figure 2: Class Imbalance and Bond Type Frequency



For buy trades (OWIC lists), Figure 3 displays the KDEs of several numeric features. This graph demonstrates that while there may be some separation between the two classes, the decision boundary is likely to be complex. This is further supported by the fact that the decision-making process of traders is not easily captured by a set of heuristic rules, as demonstrated by the lack of adoption of the explicitly programmed model. Both of these phenomena are explored further in the Methods section of this paper.

Figure 3: KDEs of Numeric Features (Buy Trades)



6 Methods

Two opposing dynamics guided the experimentation and model development efforts: the desire to train a highly performant model to replace the existing explicitly programmed model and the unique requirements of institutional trade execution workflows. While the development of a reliable, fully autonomous model capable of making trading decisions in production without the need for trader supervision would serve the overarching goal of increasing scalability, the financially sensitive nature of the modelled decision-making process (as well as the fiduciary responsibility of buy-side traders in particular) introduces an unwavering mandate for error-free execution decisions that a human trader would both agree with and be able to justify. The following sections demonstrate how these two priorities were successfully balanced through a combination of three techniques:

- Domain-tailored evaluation of model performance (Section 6.1)
- Rules-based pre- and post-processing (Section 6.2 and 6.4 respectively)
- System design decisions made when designing the software architecture surrounding the model (Section 9).

6.1 Measuring Model Performance

6.1.1 Naive Model

A direct approach to evaluating a model's performance is by gauging its accuracy:

- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
  - Overall percentage of correct model predictions

However, as displayed by Figure 2, the imbalanced distribution of the target in favor of the positive class makes it so a "model" that simply predicts EXECUTE for every trade would show as having an impressive 95% accuracy for sell trades and 86% accuracy for buys. This "always EXECUTE" model (the "naive model" from here forward) has obvious issues; false positives (predictions of EXECUTE when the trader's decision would have been PASS) carry a higher cost than false negatives (predictions of PASS when the trader's decision would have been EXECUTE) because a trade cannot be "undone" once it is executed. Clearly, accuracy alone does not capture this phenomenon, and more nuanced performance metrics must be used to complement it in finding a desirable balance between execution aggressiveness (capturing genuine executions) and conservatism (avoiding incorrect executions).

6.1.2 Explicitly Programmed Baseline Model

As noted in Section 5.2, the motivation behind developing a predictive model was to improve upon the performance of an existing explicitly programmed model (the "baseline model" from here forward) that exhibited unsatisfactory performance. The baseline model's performance for the full dataset is displayed in Figure 4. The specific flaw identified by traders was the baseline model's tendency to be overly conservative (that is, to PASS on too many items). This bias in favor of predicting PASS led to the baseline model's predictions being completely ignored by traders.

6.1.3 Domain-Specific Performance Metric Interpretations

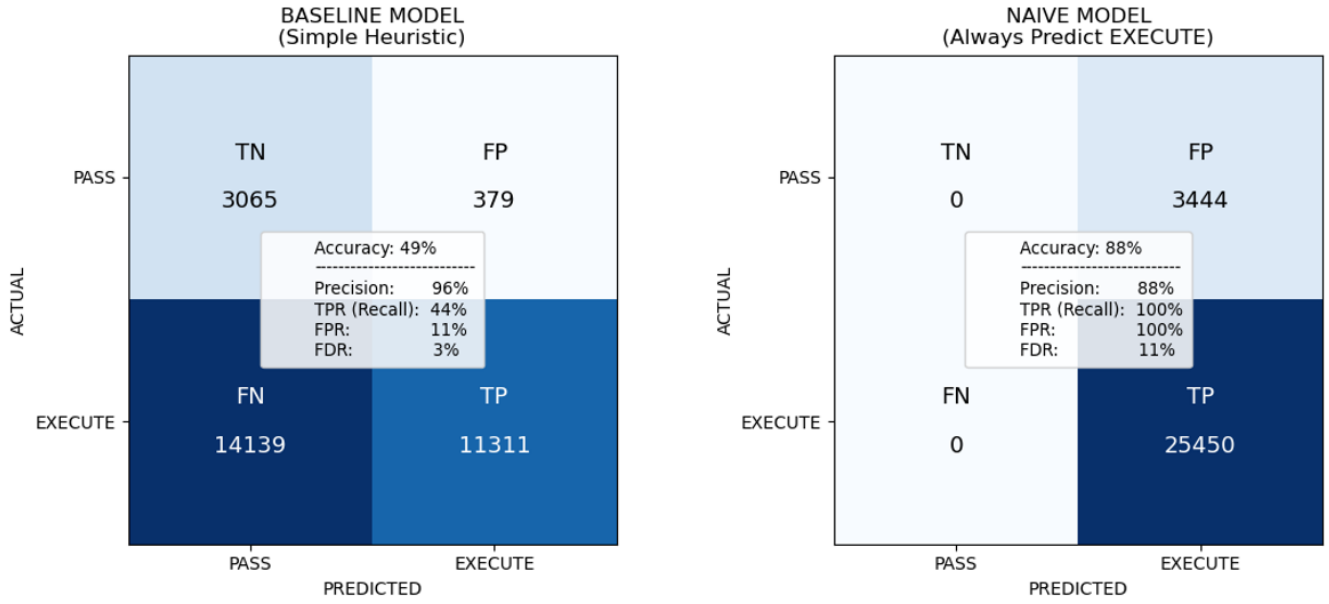
To be useful in practice, an ideal model must be able to balance the strengths of both the naive model and the baseline model by being sufficiently aggressive to capture EXECUTE decisions with high accuracy while still avoiding false positives. The below metrics provide more useful measurements of model performance than accuracy alone.

- Precision = $\frac{TP}{TP+FP}$
  - Percentage of predictions that were correct when the model predicted EXECUTE.
  - Higher precision indicates fewer false positives (mistaken executions), indicating good discretion when predicting EXECUTE.
  - Note that a model can achieve high precision while being overly conservative (as seen with the baseline model in Figure 4).

- False Discovery Rate (FDR) = $\frac{FP}{TP+FP} = 1 - \text{Precision}$
  - Percentage of predictions that were incorrect when the model predicted EXECUTE.
  - An elevated FDR indicates a "trigger happy" model that frequently predicts EXECUTE when it should pass.
- TPR (Recall) = $\frac{TP}{TP+FN}$
  - Percentage of actual executions where the model correctly predicted EXECUTE.
  - Note that a model can achieve high recall while being overly aggressive (as seen with the naive model in Figure 4).
- FPR = $\frac{FP}{FP+TN}$
  - Percentage of actual passes where the model incorrectly predicted EXECUTE.
  - Note that an infrequent negative class (PASS) can result in a high FPR from a small number of false positives.
- F1 Score = $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$
  - Harmonic mean of Precision and Recall, providing a balance between the two metrics.
  - A high F1 Score indicates a desirable balance between aggressiveness and conservatism.

Each of these metrics communicates unique and important information about model performance. Rather than selecting a single one, they are considered as a profile during model selection and evaluation. This decision to employ a variety of domain-meaningful performance evaluation metrics led directly to the developments described in the next section.

Figure 4: Confusion Matrix for Baseline and Naive Model - Full Dataset



6.2 Rules-Based Pre-Processing

The baseline model's excessive conservatism noted by traders can be seen in the accuracy of 49% and a TPR of 44% in Figure 4. However, while comparing the performance of the trained models and the baseline model during the initial model selection process, it became evident that the baseline model exhibits extremely high precision (low FDR) on both a relative and absolute basis. This performance profile indicates that the baseline model isn't nearly aggressive enough in predicting EXECUTE, but when it does predict EXECUTE, it is almost always correct. A review of the small number of false positives of the baseline model was undertaken, and they were determined to be noise (trades that should have been executed but for some reason were not). After speaking with the desk and documenting the small set of heuristic rules used by the baseline model, a policy was adopted to automatically execute ("auto-ex") the items that the baseline model predicts EXECUTE for.

This finding and the adopted policy should not be overlooked as a mundane data cleaning detail. The baseline model predicted EXECUTE for 41% of the trades in our dataset, and it is reasonable to expect that around that percentage of trades will be automatically executed going forward. Though the identification and adoption of an explicitly programmed methodology may be an atypical outcome for a research project focused on ML methods, it is a major success in this study's motivating objectives of enabling scalability and increasing trader productivity using interpretable approaches with robust reliability.

Following the adoption of this policy, the focus of experimentation shifted to the remaining subset of 17,204 trades for which the explicitly programmed model predicts PASS. Model training, selection, and evaluation were re-run from scratch using this reduced dataset (rather than the pre-reduction dataset), seeing as the reduced dataset represents the subset of trades to which the model will be applied in the final-state production workflow.

6.3 Experiment Design and Model Selection

The tested model architectures align with those discussed in Section 3: Logistic Regression, K-Nearest Neighbors, Decision Tree, Random Forest, XGBoost, CatBoost, and Multi-Layer Perceptron. To begin the model selection process, a 70-10-20 stratified train-validation-test split was performed on the full dataset. A stratified five-fold cross-validated hyperparameter grid search was conducted for each architecture. The hyperparameter search spaces are included in Appendix E. The best-performing model from the hyperparameter search was selected.

Before evaluation of this best-performing model, prediction on the validation set from the original 70-10-20 split was used for analyzing different probability threshold levels. Although the default threshold setting of 0.50 was ultimately kept in place for the final model, this process was critical to the rules-based post-processing methodology discussed next.

6.4 Rules-Based Post-Processing

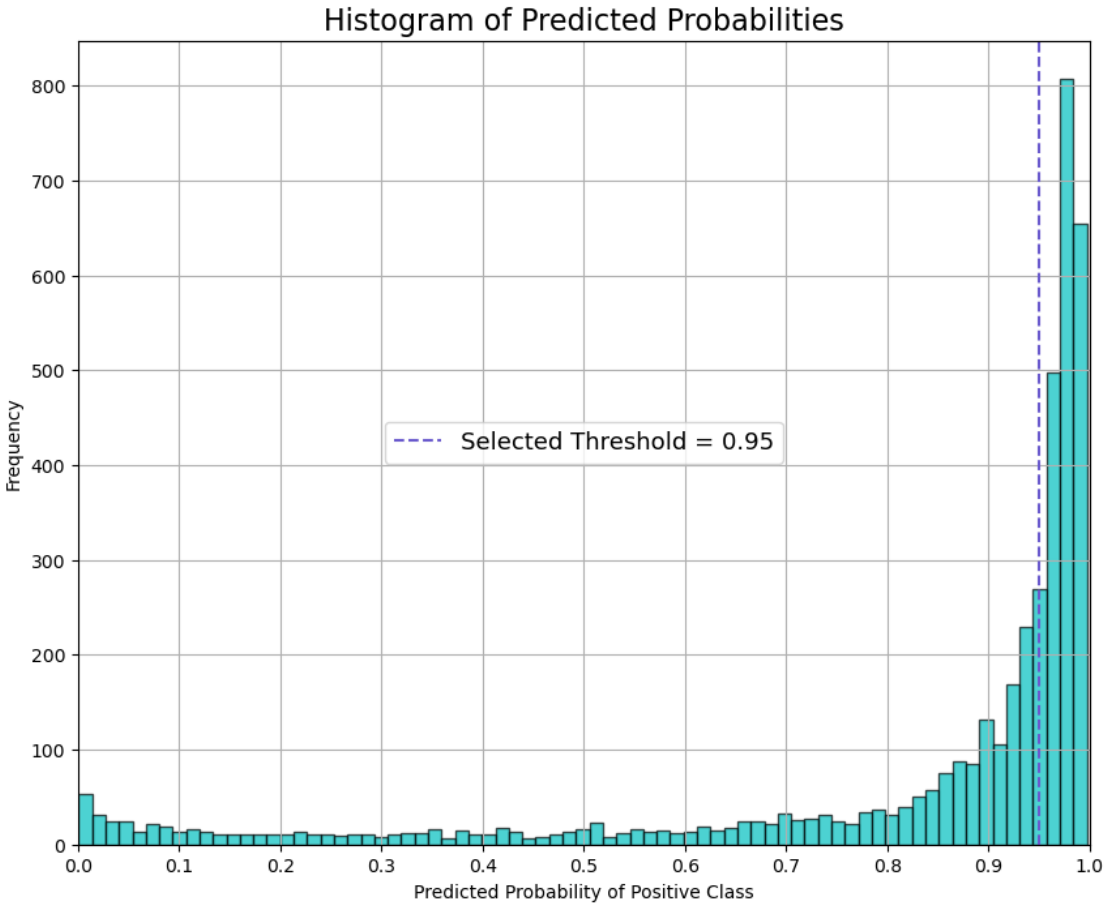
Given the high-stakes environment of the trading desk, while the predictive model showcases robust performance, a fail-safe mechanism is essential to guarantee trading decisions' fidelity. One way to achieve this is by implementing 'guardrails' around the model's predictions. This approach involves two main components:

1. A stringent probability threshold of 0.95, which identifies trades that the model predicted with extremely high confidence. This high confidence subset of trades is then subjected to a series of heuristic "guardrails." If the trades pass these checks, such as ensuring they don't

violate a specified maximum loss size, they are automatically executed.

2. For trades that do not qualify for automatic execution based on either the explicitly programmed rules or the aforementioned post-processing logic, the software architecture preselects the model-predicted decision within the trading UI. This enables traders to swiftly review, possibly amend the model's decision, and then execute all trades with a single click. The rationale and detailed design behind this approach will be elaborated upon in the System Design section.

Figure 5: Winning CatBoost Model's Predicted EXECUTE Probabilities



The selection of the 0.95 probability threshold is detailed in Section 7.2.

6.5 Comprehensive Workflow Summary

By combining the model's predictive capabilities with rules-based checks, we cultivate a synergistic approach that maximizes both efficiency and reliability.

1. Explicit Rule-Based Auto-Execution: Using the explicitly programmed baseline model, trades that meet specific criteria are auto-executed without needing further review.

2. Predictive Model Evaluation: For the remaining trades, the trained predictive model evaluates each trade's likelihood of being executed.

3. Rules-Based Post-Processing: High-confidence trades from the predictive model undergo a secondary check against heuristic guardrails. Those that pass this vetting are also auto-executed.

4. Trader Review: The remaining trades are presented to traders in the UI, with the model's recommended action pre-selected. Traders can quickly review, adjust if necessary, and execute.

This workflow, designed to be both rigorous and efficient, represents a hybrid approach that leverages the strengths of machine learning models while respecting the intricacies and demands of the trading environment. The associated data flow is depicted in Figure 9.

7 Results

7.1 Model Selection

For each tested architecture, the combination of hyperparameter settings with the highest average F1 score across folds was selected. These average F1 scores of the top-performing models were then ranked. The ranking of the models in this cross-validation-based F-Measure matched the order depicted in Figure 6 (which displays the performance of these same top models evaluated on the test set).

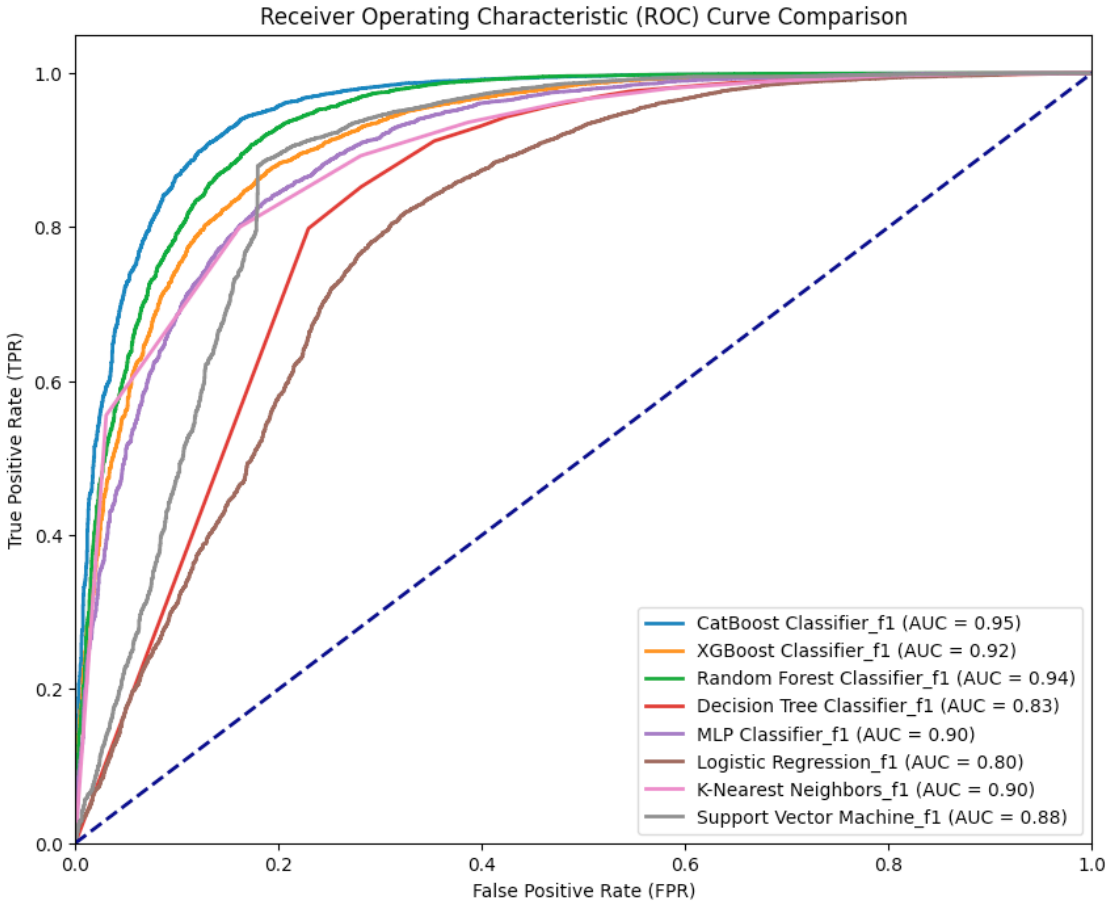
7.2 Model Evaluation

Notably, CatBoost was the top-performing model across Accuracy, Precision, and F1 Score in both the cross validation ranking and the test set evaluation and was thus selected as the architecture to move forward with. This consistency in performance is a strong indicator of the model's robustness.

Figure 6: Best Tuned Models for each Architecture vs. Reduced Test Set

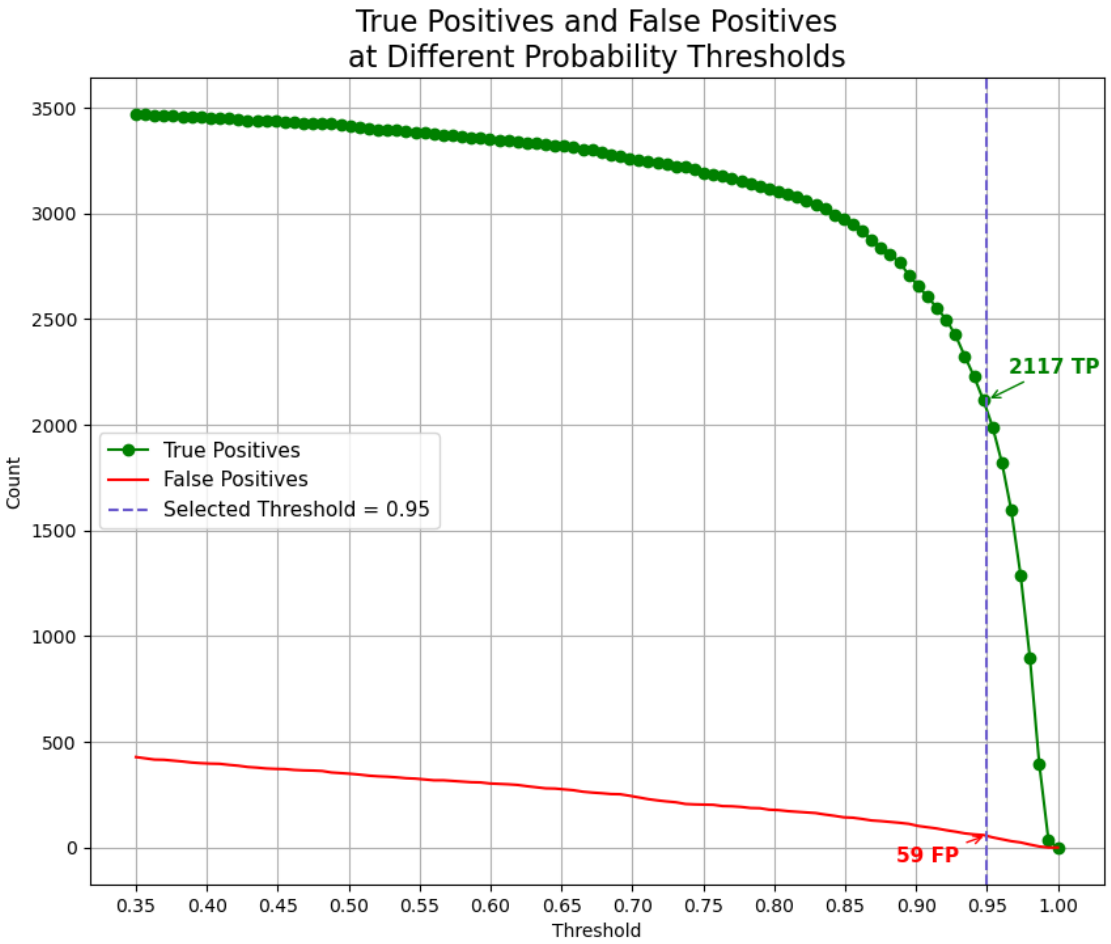
Model	TPR	FPR	FDR	Accuracy	Precision	F1
CatBoost	0.9658	0.4569	***0.09299***	***0.8905***	***0.9070***	***0.9355***
Random Forest	0.9663	0.4752	0.09630	0.8877	0.9037	0.9340
XGBoost	0.9635	0.4713	0.09583	0.8861	0.9042	0.9329
MLP	0.9621	0.4661	0.09500	0.8858	0.9050	0.9327
Decision Tree	0.9714	0.5574	0.1106	0.8772	0.8894	0.9286
K-Nearest Neighbors	0.9570	0.5144	0.1043	0.8730	0.8957	0.9253
Logistic Regression	0.9805	0.6645	0.1280	0.8656	0.8720	0.9230

Figure 7: ROC Curves of the Winnning Tuned Models on the Test Set



A custom (financial) loss-based metric was combined with a cross validation approach to select the 0.95 probability threshold. This high threshold is not the optimal probability threshold associated with any performance metric. It is intended to be highly conservative, as it is one determinant of whether a trade is sent for execution without trader review. For trades that do not meet this high bar and are sent for trader review, the standard 0.50 probability threshold is used as it shows strong balanced performance across all relevant metrics.

Figure 8: Threshold Selection



Comparing to the Naive Model and Baseline Model benchmarks from Figure 4 provides useful information regarding model efficacy in achieving the goals of balancing conservatism and aggression in execution decisions. During the first month of production deployment (accounting for ~5,500 trades reviewed), the CatBoost model has demonstrated the following performance:

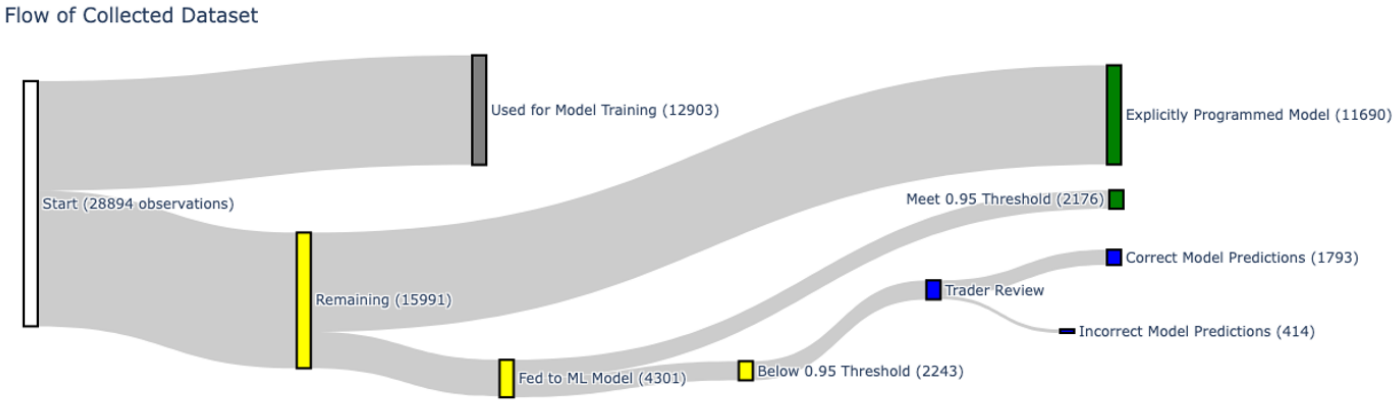
- Accuracy: 95% (vs. 49% Baseline Model, 88% Naive Model)
- Precision: 98% (vs. 96% Baseline Model, 88% Naive Model)
- TPR (Recall): 97% (vs. 44% Baseline Model, 100% Naive Model)
- FPR: 32% (vs. 11% Baseline Model, 100% Naive Model)
- FDR: 2% (vs. 3% Baseline Model, 11% Naive Model)

The trained model compares favorably to both models and performance is further improved when the "hybrid model" depicted in Figure 9 is accounted for.

8 Analysis

8.1 Workflow Design and Economic Impact

Figure 9: Sankey Diagram of Collected Trade Data



It is worth noting that Figures 7 and 8 capture only the models' performance on the post-reduction dataset, not the performance of the system considered as a whole. The bottom branch in Figure 9 illustrates the performance of the end-to-end execution process *for the collected dataset only* (pre-deployment). Green is used in the Sankey diagram to signify trading decisions that are automatedly sent for execution based on the hybrid expert systems + high-confidence ML prediction branches. Blue is used to signify the predictions made by the CatBoost model that do not meet the high-confidence probability threshold and thus require trader review. Of the 15,991 trades available for evaluation after the training set is removed, the system automatedly executes 86% of these trades with high reliability and transparency. Of the 2,207 trades that are highlighted for trader review, the model predicts 1,793 correctly. The remaining 414 trades highlight trades for which the trader either actively disagrees with the model's prediction or extenuating circumstances were involved.

The proposed hybrid auto-ex system mirrors the bottom branch in Figure 9 (and delineated in the Comprehensive Workflow Summary discussion in section 6.5). Currently the desk is in a user acceptance phase for the proposed execution workflow, with all trades being reviewed by traders to ensure stability of model predictions in the context of surrounding trading workflows. Given current trends of the desk and usage of the system (the first month of deployment resulting in the system being used to execute $300 million in trade volume across 5,500 reviewed trades), the system as a whole is expected to execute $3.6 billion worth of trading volume between its introduction in July 2023 and July of 2024.

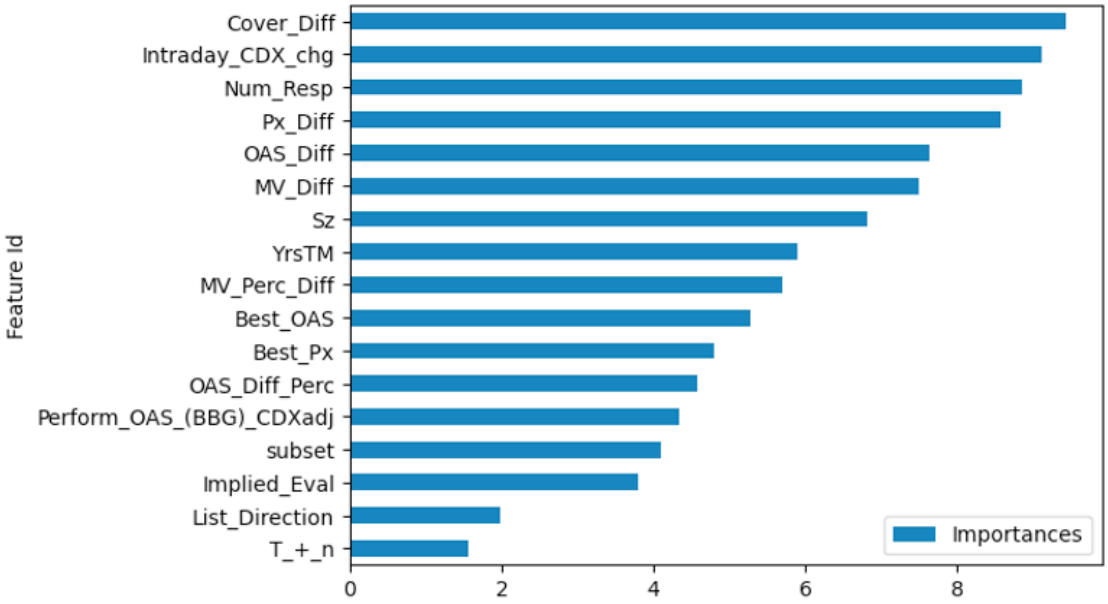
As Figure 9 and the preceding sections illustrate, achieving trust and transparency in the execution workflow is paramount for the trading desk. This transparency not only stems from the model's predictive capabilities but also from the inherent logic embedded in the system. The core CatBoost model of our system is gauged by its capability to segment the feature space, maximizing information gain or minimizing entropy with each division. Our workflow design similarly aimed to identify key decision junctions in the trading process. For instance, the explicit rule-based auto-execution and the post-processing checks prior to auto-execution can be viewed as systematic efforts to identify decision boundaries that can be used to ensure model behavior is aligned with trader

expectations.

8.2 Model Interpretability

GBDTs, such as CatBoost, have faced criticism for occasionally lacking clarity and explainability. Though the ranked feature importance of the CatBoost model would be sensible to traders (the top ten in descending order being OAS_Diff, Px_Diff, OAS_Diff_Perc, MV_Perc_Diff, MV_Diff, Num_Resp, Sz, Best_OAS, YrsTM, and Cover_Diff), the feature importances of XGBoost model are displayed in Figure 10. Notably, Figure 10 ranks 'Cover_Diff' and 'Num_Resp'—potentially seen as only marginally relevant by bond traders—among the top three in feature importance.

Figure 10: Feature Importances for best XGBoost Model

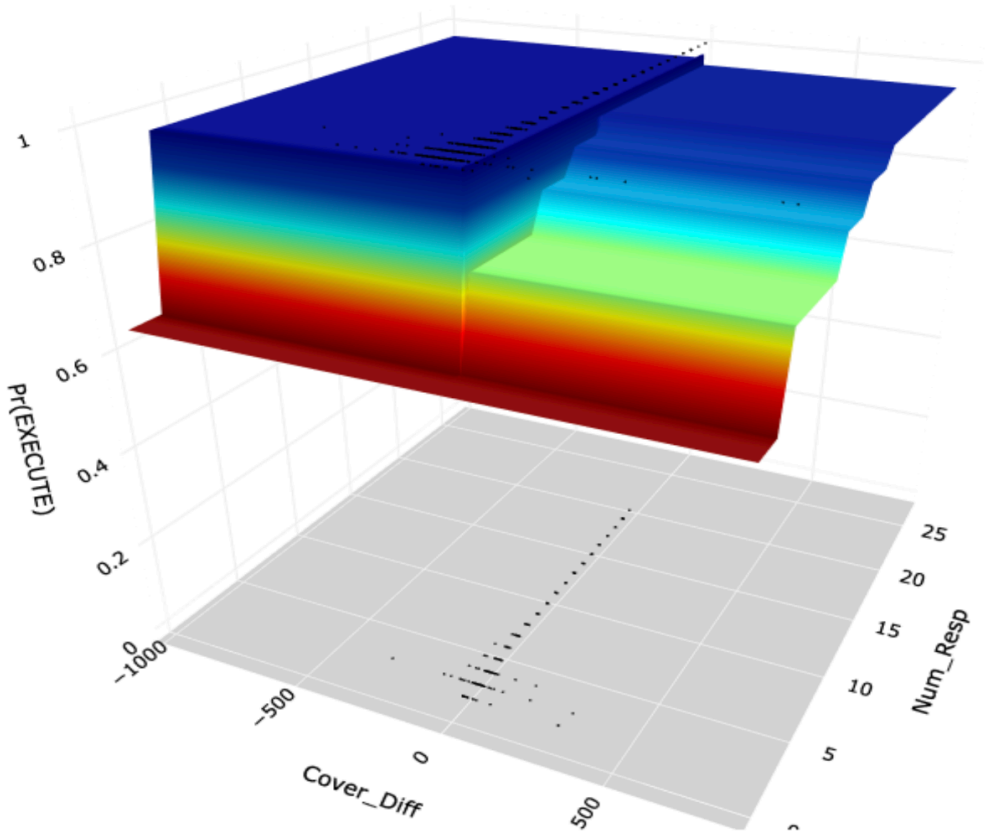


This underscores the idea that ML, particularly in the realm of financial markets, involves significant empirical experimentation. In the process of representing high-dimensional, non-linear relationships, model behavior that might appear orthogonal to domain expertise can arise. Attempting to force the model to adhere to preconceived mental representations of relationships, while appropriate in some circumstances, may degrade model performance. Early modeling efforts for this project attempted to create distinct models for buy trades, sell trades, and even models trained specifically for inference on buy and sales for each individual class of bond in the dataset (e.g. fixed rates, floating rate notes, agencies, etc.). All of those methods consistently underperformed the final CatBoost model that modeled all trade and bond types in a single model.

Efforts to understand model internals like feature importance are nonetheless a valuable practice. Keeping in mind that decision trees repeatedly search for feature thresholds to make class prediction splits, note that in Figure 3 in Section 5.4 Cover_Diff and Num_Resp are two features for which clean separations of the positive class exist. Figure 11 further demonstrates this point. When fitting and surface plotting a decision tree on just the two aforementioned features (for the sake of demonstration/analysis only), a discernible pattern emerges. The model learns the

Cover_Diff feature encodes information about trade direction, splitting that feature precisely at zero (explaining why List_Direction is a low importance feature across models). Additionally, it uncovers a propensity to execute trades only rarely when the number of responses is below three, regardless of trade direction. As the number of responses begins to rise, the model learns a more complex profile for buy trade behavior than for sales—a logical pattern, given the execution rates shown in the left side of Figure 2 and, more concretely, bond traders' tendency to be more selective when buying relative to selling (as the selling is often driven by non-discretionary activity).

Figure 11: Modelled Pr(EXECUTE) from a Decision Tree fit on Cover Diff and Num Resp Features



However, the proficiency of ML models to unearth such patterns can be a double-edged sword. Early modeling efforts in this project revealed the predictive power of using the time of day as a feature. A closer examination revealed that this relationship was due to traders' (including the author) tendency to queue large lists right at market close and run out of time before being able to execute, resulting in a disproportionate number of passed trades within a specific timeframe. While beneficial to model performance, such features are not conducive for incorporation in an execution policy and underscore the importance of model inspection, pre-execution review by traders, and the guardrails built into the proposed workflow.

9 Discussion

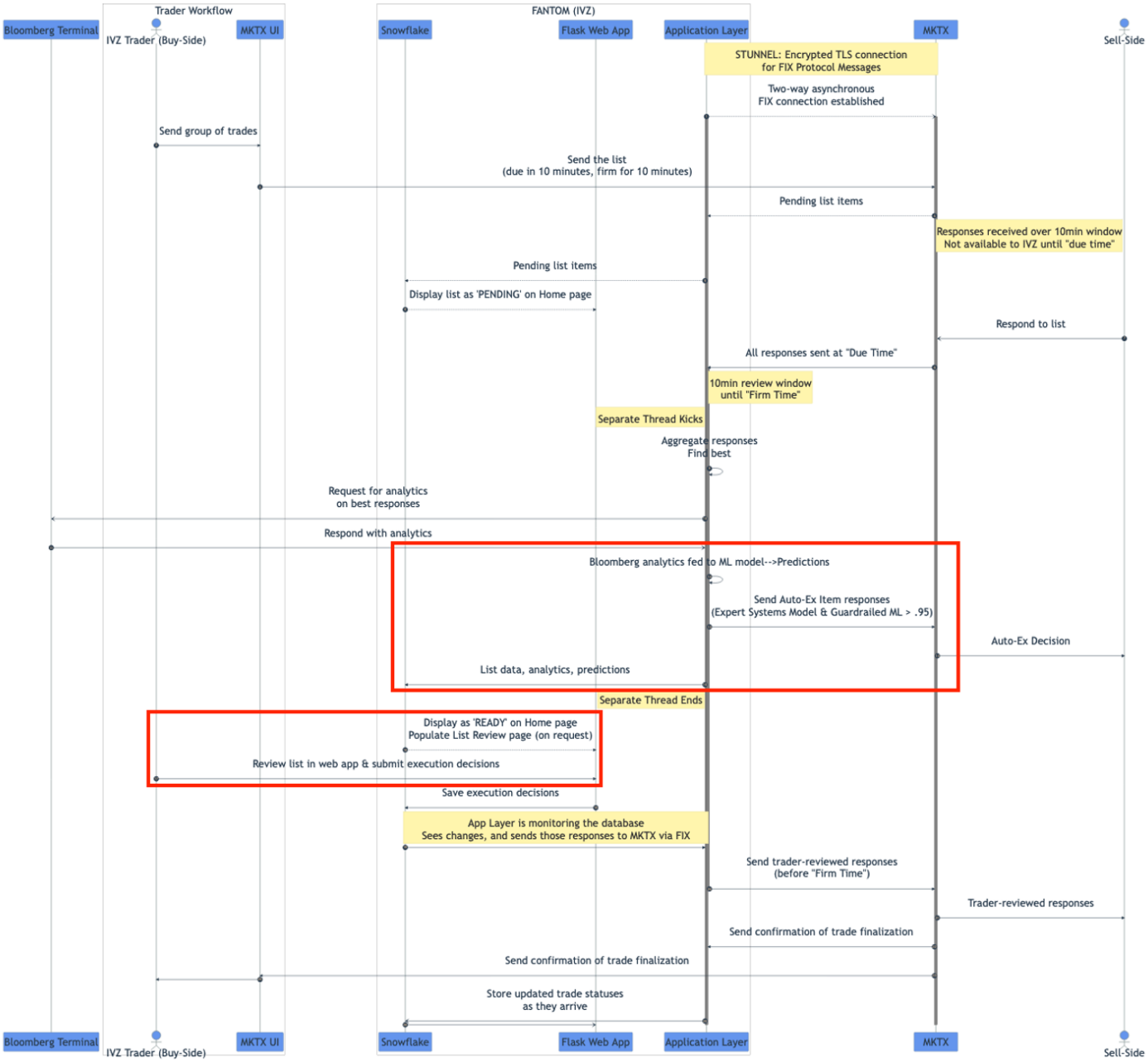
9.1 System Design

The trained CatBoost model sits within FANTOM (Flow Aggregator aNd Trade OptiMizer), an integrated real-time ML inference and trade execution system. FANTOM continuously monitors and interacts with the MarketAxess platform using a FIX Protocol (Financial Information Exchange Protocol) messaging feed. This FIX connection (a real-time point-to-point messaging channel communicating bidirectional FIX messages via a TCP/IP tunnel that persists during market hours) is pivotal not only for sending and receiving instant communication of all trading activities between MarketAxess and the Invesco desk, but also for feeding the feature creation pipeline to enable inference by the CatBoost model. A custom real-time data processing layer (the "application layer") monitors the FIX connection for incoming messages, decodes and stores selected trade information, orchestrates the calculation and retrieval of features (including asynchronous communication with the Bloomberg BPIPE and other APIs), and manages both the auto-execution flow described earlier as well as inference by the CatBoost model. On the sending side, the application layer also monitors the Snowflake database to capture and instantly communicate trader execution decisions via FIX as they are made in the web UI.

Figure 12 depicts both sides of the orchestration done by FANTOM's application layer for a single list of trades. At a high level, raw FIX messages are translated to features for model inference then to trade decisions and back to FIX messages while interfacing with the Flask front-end via the Snowflake back-end and storing data for model monitoring and retraining. Though the depicted sequence presents a synchronous instance with a single trader executing one list, FANTOM must asynchronously cater to all traders (with individual traders often running several lists concurrently). This necessitates that FANTOM remains available, responsive, and scalable to cater to varying trading volumes and activities without interruptions.

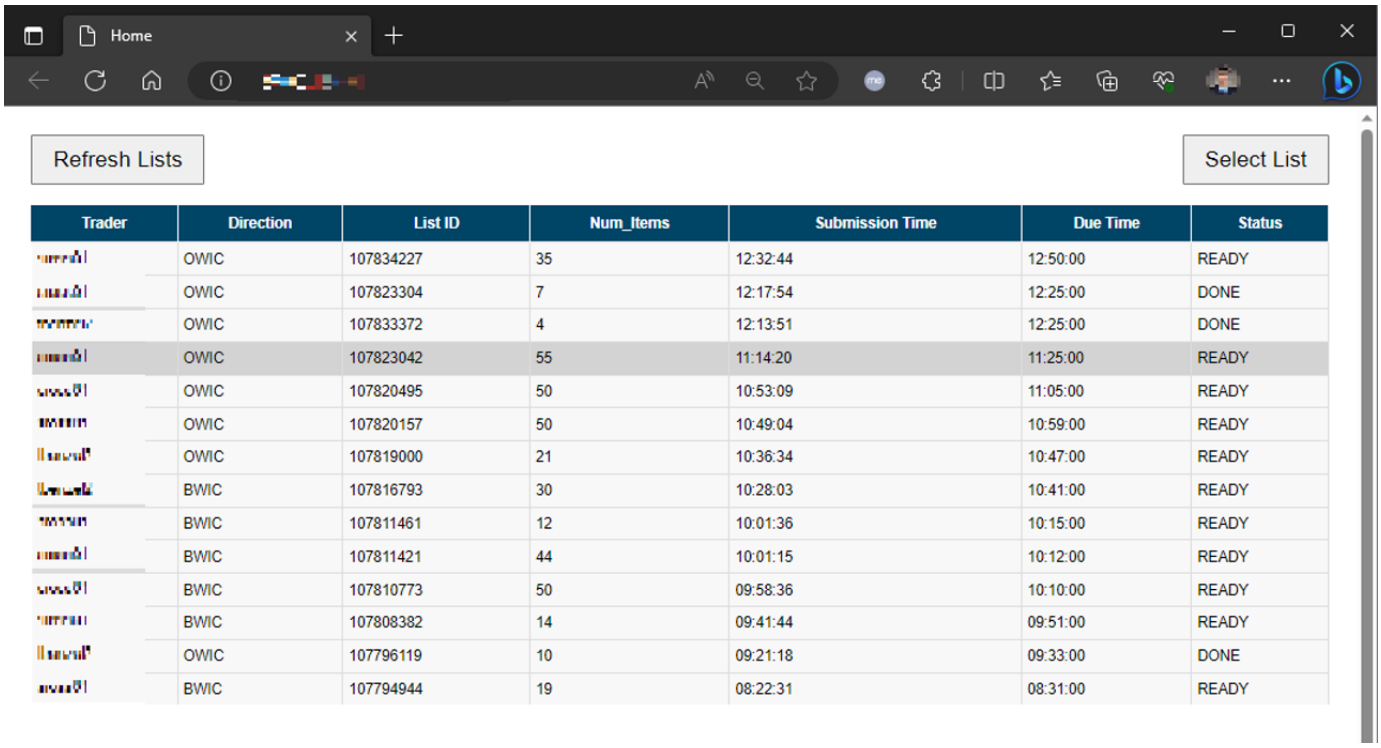
This system has been in production for approximately a month at the time of writing. The application layer resides on an on-premise Invesco server, while the Flask app operates on Amazon's Elastic Kubernetes Service (AWS EKS). FANTOM interfaces closely with the QuickFix library (a C++ library akin to Apache Kafka but specialized for trading-specific use cases) for low-level FIX protocol orchestration and utilizes Python, SQL, and Javascript for core functionality.

Figure 12: System Architecture Diagram Showing Trading Workflow and Real-Time Model Integration



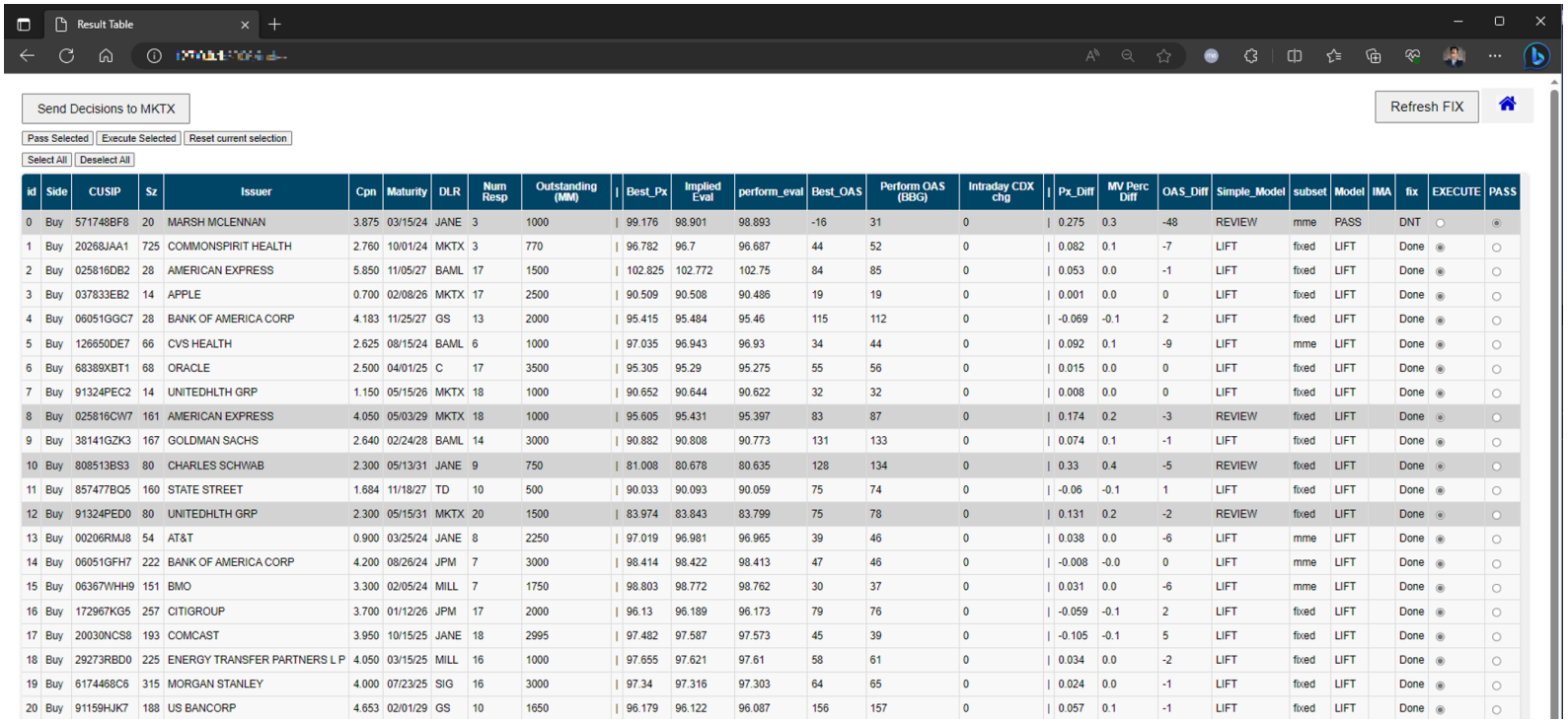
Figures 13 and 14 display FANTOM's Flask-based web UI. Traders will recognize this as a browser-based replication of the MarketAxess workstation with enhanced functionality (additional analytics and model prediction). Figure 14 displays how the UI auto-populates trade decisions in accordance with model predictions so that traders can execute full lists in a single click.

Figure 13: Screenshot of the FANTOM (Flow Aggregator aNd Trade OptiMizer) Home Page



Trader	Direction	List ID	Num_Items	Submission Time	Due Time	Status
[redacted]	OWIC	107834227	35	12:32:44	12:50:00	READY
[redacted]	OWIC	107823304	7	12:17:54	12:25:00	DONE
[redacted]	OWIC	107833372	4	12:13:51	12:25:00	DONE
[redacted]	OWIC	107823042	55	11:14:20	11:25:00	READY
[redacted]	OWIC	107820495	50	10:53:09	11:05:00	READY
[redacted]	OWIC	107820157	50	10:49:04	10:59:00	READY
[redacted]	OWIC	107819000	21	10:36:34	10:47:00	READY
[redacted]	BWIC	107816793	30	10:28:03	10:41:00	READY
[redacted]	BWIC	107811461	12	10:01:36	10:15:00	READY
[redacted]	BWIC	107811421	44	10:01:15	10:12:00	READY
[redacted]	BWIC	107810773	50	09:58:36	10:10:00	READY
[redacted]	BWIC	107808382	14	09:41:44	09:51:00	READY
[redacted]	OWIC	107796119	10	09:21:18	09:33:00	DONE
[redacted]	BWIC	107794944	19	08:22:31	08:31:00	READY

Figure 14: Screenshot of the FANTOM Model Prediction/Trader Review/Execution Web Page



id	Side	CUSIP	Sz	Issuer	Cpn	Maturity	DLR	Num Resp	Outstanding (MM)	Best_Px	Implied Eval	perform_eval	Best_OAS	Perform OAS (BBG)	Intraday CDX chg	Px_Diff	MV Perc Diff	OAS_Diff	Simple_Model	subset	Model	IMA	fix	EXECUTE	PASS
0	Buy	571748BF8	20	MARSH MCLENNAN	3.875	03/15/24	JANE	3	1000	99.176	98.901	98.893	-16	31	0	0.275	0.3	-48	REVIEW	mme	PASS		DNT		
1	Buy	20268JAA1	725	COMMONSPIRIT HEALTH	2.760	10/01/24	MKTX	3	770	96.782	96.7	96.687	44	52	0	0.082	0.1	-7	LIFT	fixed	LIFT		Done		
2	Buy	025816DB2	28	AMERICAN EXPRESS	5.850	11/05/27	BAML	17	1500	102.825	102.772	102.75	84	85	0	0.053	0.0	-1	LIFT	fixed	LIFT		Done		
3	Buy	037833EB2	14	APPLE	0.700	02/08/26	MKTX	17	2500	90.509	90.508	90.486	19	19	0	0.001	0.0	0	LIFT	fixed	LIFT		Done		
4	Buy	06051GGC7	28	BANK OF AMERICA CORP	4.183	11/25/27	GS	13	2000	95.415	95.484	95.46	115	112	0	-0.069	-0.1	2	LIFT	fixed	LIFT		Done		
5	Buy	126650DE7	66	CVS HEALTH	2.625	08/15/24	BAML	6	1000	97.035	96.943	96.93	34	44	0	0.092	0.1	-9	LIFT	mme	LIFT		Done		
6	Buy	68389XBT1	68	ORACLE	2.500	04/01/25	C	17	3500	95.305	95.29	95.275	55	56	0	0.015	0.0	0	LIFT	fixed	LIFT		Done		
7	Buy	91324PEC2	14	UNITEDHLTH GRP	1.150	05/15/26	MKTX	18	1000	90.652	90.644	90.622	32	32	0	0.008	0.0	0	LIFT	fixed	LIFT		Done		
8	Buy	025816CW7	161	AMERICAN EXPRESS	4.050	05/03/29	MKTX	18	1000	95.605	95.431	95.397	83	87	0	0.174	0.2	-3	REVIEW	fixed	LIFT		Done		
9	Buy	38141GZK3	167	GOLDMAN SACHS	2.640	02/24/28	BAML	14	3000	90.882	90.808	90.773	131	133	0	0.074	0.1	-1	LIFT	fixed	LIFT		Done		
10	Buy	808513BS3	80	CHARLES SCHWAB	2.300	05/13/31	JANE	9	750	81.008	80.678	80.635	128	134	0	0.33	0.4	-5	REVIEW	fixed	LIFT		Done		
11	Buy	857477BQ5	160	STATE STREET	1.684	11/18/27	TD	10	500	90.033	90.093	90.059	75	74	0	-0.06	-0.1	1	LIFT	fixed	LIFT		Done		
12	Buy	91324PED0	80	UNITEDHLTH GRP	2.300	05/15/31	MKTX	20	1500	83.974	83.843	83.799	75	78	0	0.131	0.2	-2	REVIEW	fixed	LIFT		Done		
13	Buy	00206RMJ8	54	AT&T	0.900	03/25/24	JANE	8	2250	97.019	96.981	96.965	39	46	0	0.038	0.0	-6	LIFT	mme	LIFT		Done		
14	Buy	06051GFH7	222	BANK OF AMERICA CORP	4.200	08/26/24	JPM	7	3000	98.414	98.422	98.413	47	46	0	-0.008	-0.0	0	LIFT	mme	LIFT		Done		
15	Buy	06367WHH9	151	BMO	3.300	02/05/24	MILL	7	1750	98.803	98.772	98.762	30	37	0	0.031	0.0	-6	LIFT	mme	LIFT		Done		
16	Buy	172967KG5	257	CITIGROUP	3.700	01/12/26	JPM	17	2000	96.13	96.189	96.173	79	76	0	-0.059	-0.1	2	LIFT	fixed	LIFT		Done		
17	Buy	20030NCS8	193	COMCAST	3.950	10/15/25	JANE	18	2995	97.482	97.587	97.573	45	39	0	-0.105	-0.1	5	LIFT	fixed	LIFT		Done		
18	Buy	29273RBD0	225	ENERGY TRANSFER PARTNERS L P	4.050	03/15/25	MILL	16	1000	97.655	97.621	97.61	58	61	0	0.034	0.0	-2	LIFT	fixed	LIFT		Done		
19	Buy	6174468C6	315	MORGAN STANLEY	4.000	07/23/25	SIG	16	3000	97.34	97.316	97.303	64	65	0	0.024	0.0	-1	LIFT	fixed	LIFT		Done		
20	Buy	91159HJK7	188	US BANCORP	4.653	02/01/29	GS	10	1650	96.179	96.122	96.087	156	157	0	0.057	0.1	-1	LIFT	fixed	LIFT		Done		

Undertaking FANTOM presented a multifaceted challenge, intertwining both engineering and computer science complexities. The high throughput nature of the messaging system (with functional requirements including scaling to process thousands of FIX messages per second) required careful design decisions to successfully embed real-time processing and inference in a manner that was simultaneously highly robust, efficient, and performant. It's noteworthy that although model development was crucial, it represented a mere 10% of the overall project effort. The remaining 90% of the effort was devoted to engineering and system design, highlighting the significance of system architecture in deploying ML models in dynamic settings like trading.

10 Future Work

This paper lays some initial groundwork for further exploration of financial machine learning in the fixed income domain. Several avenues are of particular interest:

1. Enhanced Modeling Techniques: While CatBoost has proven effective, deep learning and reinforcement learning approaches were left unexplored in this context. Although a Wide & Deep and TabFormer architectures were initially considered, their performance did not surpass the best-performing MLP and thus were not included in experimentation. However, as new transformer-based models and other deep learning variations are invented and improved upon for tabular data, integrating these approaches might offer nuanced insights and further optimize trade predictions. Additionally, although the binary classification formulation was most appropriate for our particular use case (due to the lack of dependence between individual trades), many trading use cases may find reinforcement learning or counterfactual regret minimization more fitting.

2. Feature Engineering: As FANTOM collects large amounts of data every trading day, leveraging the latent information in our increasing dataset in a way that improves model performance will be a major focus. In the first month of deployment, we have identified several features that will be incorporated in version 2 of FANTOM. Building a scalable pipeline for collecting new features and using them for real-time inference is an important area of focus, as are model monitoring and retraining pipelines.

3. User Experience and Interface Design: As the FANTOM system becomes adopted by traders, enhancing its user interface and integrating user feedback can make the tool more intuitive and efficient.

11 Conclusion

The evolution of financial markets and trading systems, combined with the rapid advancement of machine learning, indicates a rich and promising future for financial machine learning. This paper showcases the potential of integrating sophisticated algorithms within the rigorous decision-making process of an institutional buy-side fixed income desk. Our endeavors began with an exploration of modeling techniques, emphasizing the value and adaptability of the CatBoost algorithm. Building upon the foundation of a robust predictive model, the design of a hybrid auto-execution workflow was explained in the light of aiming to achieve model-driven scalability in a responsible and risk-aware way. This resulting workflow is brought to life in the FANTOM system, which has proved to be an immediately useful and reliable system that traders enjoy using.

As financial markets continue to evolve and integrate cutting-edge technologies, the possibilities that lie at the intersection of finance and machine learning are poised for explosive growth. Careful modeling, robust automation, and human expertise will be the foundations for ML's increasing presence in the industry.

12 References

Breiman, Leo. 2001. "Random Forests." Machine Learning.

Breiman, Leo, Jerome Friedman, Richard Olshen, and Charles Stone. 1986. Classification and Regression Trees. Wadsworth.

Chen, Tianqi, and Carlos Guestrin. 2016. "XGBoost: A scalable tree boosting system." In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Cover, Thomas, and Peter Hart. 1967. "Nearest neighbor pattern classification." IEEE Transactions on Information Theory.

Cox, D.R. 1958. "The Regression Analysis of Binary Sequences." Journal of the Royal Statistical Society.

Prokhorenkova, Liudmila, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2017. "CatBoost: unbiased boosting with categorical features." In Advances in Neural Information Processing Systems.

Quinlan, J.R. 1986. "Induction of decision trees." Machine Learning.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. 1986. "Learning representations by back-propagating errors." Nature.