Jane Street Market Data Forecasting

XIE ZUOYU & LI SINUAN

23 June 2025

Abstract: Project Introduction

- This project is based on the Kaggle competition: **Jane Street Real-Time Market Data Forecasting**, hosted by Jane Street Capital.
- The competition aims to forecast whether a given transaction should be executed, using real-time high-dimensional market data.
- Although we joined after the official deadline and could not submit, we obtained the full dataset via the Kaggle community for independent research.
- **Dataset overview:**
  - Over **60 million rows** of historical trading records
  - **79 anonymized features**: `feature_00` to `feature_78`
  - `responder_6` as the key target variable: indicates the weighted return
  - `symbol_id` and `time_id` for time-series alignment

Abstract: Models Used

- Our goal is to evaluate the effectiveness of different supervised learning models in predicting positive returns (`responder_6 > 0`).
- All models were trained under a consistent pipeline including:
  - Handling missing values
  - Feature selection (regularization or importance ranking)
  - Evaluation using metrics such as **AUC**, accuracy, and class-wise recall
- The four models we implemented:
  1. **Logistic Regression (LR)** – a baseline linear classifier
  2. **XGBoost** – optimized gradient boosting (focus of this talk)
  3. **LightGBM** – gradient boosting using decision trees
  4. **MLP** – multi-layer neural network

Dataset Overview

- 60+ million trading records
- 79 anonymized features: `feature_00` to `feature_78`
- Target: `responder_6` (weighted return)
- Binary classification: label = 1 if `responder_6 > 0`, else 0

Data Partition Strategy

- The dataset is divided into 10 partitions: `partition_id = 0` to `partition_id = 9`.
- Based on modeling needs, we regrouped them into five subsets:
  - **Train set:** `partition_id = 0--4` — model fitting
  - **Validation set:** `partition_id = 5` — hyperparameter tuning, early stopping
  - **Test set:** `partition_id = 6--7` — final evaluation
  - **Future input:** `partition_id = 8` — used for prediction (labels removed)
  - **Future eval:** `partition_id = 9` — ground truth for future prediction (only `responder_6`, `symbol_id`, `time_id`)

Logistic Regression: Data Preprocessing

- Removed features with high missing values (e.g., 21, 26, 27, 31)
- Imputed remaining missing values with **mean**
- Applied **L1 regularization** to test feature importance
- Final feature count: **75** retained

```
In [2]: runfile('D:/My File/SMU/QF632/Final Work/Final.py', wdir='D:/My File/SMU/QF632/Final Work')
The features with the highest missing ratios are as follows (top 20):
 1. feature_21   - Missing ratio: 48.94%
 2. feature_31   - Missing ratio: 48.94%
 3. feature_27   - Missing ratio: 48.94%
 4. feature_26   - Missing ratio: 48.94%
 5. feature_00   - Missing ratio: 18.91%
 6. feature_02   - Missing ratio: 18.91%
 7. feature_03   - Missing ratio: 18.91%
 8. feature_04   - Missing ratio: 18.91%
 9. feature_01   - Missing ratio: 18.91%
10. feature_42   - Missing ratio: 12.91%
11. feature_39   - Missing ratio: 12.91%
12. feature_50   - Missing ratio: 12.63%
13. feature_53   - Missing ratio: 12.63%
14. feature_41   - Missing ratio: 3.15%
15. feature_44   - Missing ratio: 3.15%
16. feature_52   - Missing ratio: 2.86%
17. feature_55   - Missing ratio: 2.86%
18. feature_15   - Missing ratio: 2.72%
19. feature_65   - Missing ratio: 1.86%
20. feature_46   - Missing ratio: 1.86%
```

```
In [1]: runfile('D:/My File/SMU/QF632/Final Work/Final.py', wdir='D:/My File/SMU/QF632/Final Work')
The number of features retained after L1 regularization: 75
 1: feature_00
 2: feature_01
 3: feature_02
 4: feature_03
 5: feature_04
 6: feature_05
 7: feature_06
 8: feature_07
 9: feature_08
10: feature_09
11: feature_10
12: feature_11
13: feature_12
14: feature_13
15: feature_14
16: feature_15
17: feature_16
18: feature_17
19: feature_18
20: feature_19
conda: base (Python 3.12.7)   Completions: conda(base)   LSP: Pyth   ol 4   UTF-8   CRLF   RW
```

Logistic Regression: Model Performance

- Applied L2-regularized Logistic Regression using **75** features after advanced cleaning.
- Training based on partition id 0–4, validation on partition id 5.
- **Results:**
- Val AUC: `0.5596`, Test AUC: `0.5511`
- Class 1 recall: `0.3550` — room for improvement

```
In [1]: runfile('D:/My File/SMU/QF632/Final Work/Final.py', wdir='D:/My File/SMU/QF632/Final
Work')
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5627    0.7241    0.6333   2856415
           1     0.5289    0.3550    0.4248   2491785

    accuracy                         0.5521   5348200
   macro avg     0.5458    0.5396    0.5291   5348200
weighted avg     0.5470    0.5521    0.5362   5348200

[Val] AUC: 0.559619424196808
```

Logistic Regression: Validation Visualizations

- ROC curve and Confusion Matrix on validation set
- Evaluate model discriminative power and error distribution

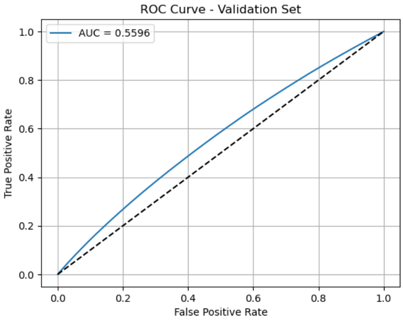


Figure 4: ROC Curve

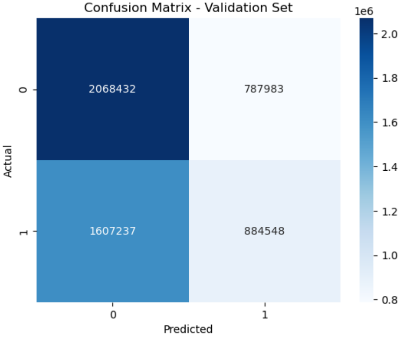


Figure 5: Confusion Matrix

Logistic Regression: Summary

- Logistic Regression, despite its simplicity, performs robustly on a high-dimensional and large-scale dataset.
- However, it struggles to capture complex nonlinear patterns, resulting in limited performance on key metrics:
  - Validation AUC: **0.5596**
  - Class 1 Recall: **0.3550**
- The model's limited recall for profitable trades motivates the exploration of more powerful nonlinear methods.
- **Therefore, we next turn to XGBoost to investigate whether tree-based models can improve predictive performance.**

XGBoost: Feature Importance

- Applied `XGBClassifier` with `logloss` as the objective function.
- All 75 features showed non-zero importance, indicating broad feature utility.
- Based on feature importance scores, we evaluated three subsets:
  - 75 features: full set
  - 65 features: importance score ≥ 0.006
  - 28 features: importance score > 0.01
- Feature selection aimed to reduce complexity while maintaining performance.

```
 bst.update(dtrain, iteration=i, fobj=obj)
Top 30 most important features (by XGBoost):
 1. feature_06   - importance: 0.1102
 2. feature_60   - importance: 0.0682
 3. feature_59   - importance: 0.0605
 4. feature_07   - importance: 0.0464
 5. feature_54   - importance: 0.0325
 6. feature_51   - importance: 0.0289
 7. feature_68   - importance: 0.0240
 8. feature_04   - importance: 0.0206
 9. feature_30   - importance: 0.0189
10. feature_55   - importance: 0.0177
11. feature_75   - importance: 0.0171
12. feature_52   - importance: 0.0150
13. feature_15   - importance: 0.0146
14. feature_71   - importance: 0.0143
15. feature_69   - importance: 0.0140
16. feature_14   - importance: 0.0128
17. feature_50   - importance: 0.0127
18. feature_01   - importance: 0.0124
19. feature_02   - importance: 0.0123
20. feature_53   - importance: 0.0123
21. feature_56   - importance: 0.0123
22. feature_58   - importance: 0.0116
23. feature_48   - importance: 0.0114
24. feature_36   - importance: 0.0107
25. feature_10   - importance: 0.0107
26. feature_23   - importance: 0.0106
27. feature_76   - importance: 0.0100
28. feature_19   - importance: 0.0100
29. feature_18   - importance: 0.0099
30. feature_29   - importance: 0.0097
```

```
31. feature_05   - importance: 0.0095
32. feature_17   - importance: 0.0094
33. feature_47   - importance: 0.0093
34. feature_66   - importance: 0.0092
35. feature_09   - importance: 0.0091
36. feature_00   - importance: 0.0091
37. feature_72   - importance: 0.0090
38. feature_22   - importance: 0.0089
39. feature_11   - importance: 0.0088
40. feature_20   - importance: 0.0086
41. feature_73   - importance: 0.0085
42. feature_61   - importance: 0.0084
43. feature_28   - importance: 0.0082
44. feature_08   - importance: 0.0081
45. feature_40   - importance: 0.0080
46. feature_13   - importance: 0.0079
47. feature_44   - importance: 0.0079
48. feature_24   - importance: 0.0079
49. feature_65   - importance: 0.0077
50. feature_67   - importance: 0.0075
51. feature_37   - importance: 0.0074
52. feature_03   - importance: 0.0073
53. feature_49   - importance: 0.0073
54. feature_38   - importance: 0.0072
55. feature_25   - importance: 0.0070
56. feature_39   - importance: 0.0070
57. feature_78   - importance: 0.0070
58. feature_70   - importance: 0.0068
59. feature_46   - importance: 0.0068
60. feature_12   - importance: 0.0067
61. feature_41   - importance: 0.0067
62. feature_62   - importance: 0.0066
63. feature_57   - importance: 0.0065
64. feature_42   - importance: 0.0062
65. feature_45   - importance: 0.0061
66. feature_33   - importance: 0.0059
67. feature_74   - importance: 0.0059
68. feature_32   - importance: 0.0058
69. feature_35   - importance: 0.0058
70. feature_34   - importance: 0.0057
71. feature_43   - importance: 0.0057
72. feature_64   - importance: 0.0055
73. feature_77   - importance: 0.0055
74. feature_63   - importance: 0.0047
75. feature_16   - importance: 0.0036
```

XGBoost: Prediction with 75 Features

```
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5783    0.6479    0.6111   2856415
           1     0.5318    0.4584    0.4924   2491785

    accuracy                         0.5596   5348200
   macro avg     0.5550    0.5531    0.5517   5348200
weighted avg     0.5566    0.5596    0.5558   5348200

[Val] AUC: 0.5760809452025897
[TEST] AUC: 0.561606945142231
```

- Improved recall and AUC compared to LR
- Better alignment with actual market directions

XGBoost: Prediction with 65 Features

```
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5785    0.6472    0.6109   2856415
           1     0.5318    0.4594    0.4930   2491785

    accuracy                         0.5597   5348200
   macro avg     0.5551    0.5533    0.5519   5348200
weighted avg     0.5567    0.5597    0.5559   5348200

[Val] AUC: 0.5761781268468457
[TEST] AUC: 0.5624955605090792
```

- Best overall AUC and accuracy among all settings
- Efficient and interpretable model

XGBoost: Prediction with 28 Features

```
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5779    0.6497    0.6117   2856415
           1     0.5317    0.4560    0.4910   2491785

    accuracy                         0.5594   5348200
   macro avg     0.5548    0.5528    0.5513   5348200
weighted avg     0.5564    0.5594    0.5554   5348200

[Val] AUC: 0.5756430997223801
[TEST] AUC: 0.5623682031790292
```

- Slight performance drop in recall and AUC
- Over-simplification may remove predictive signals

XGBoost: Prediction vs Future Data

XGBoost: Visual Analysis Summary

Feature Count	Val AUC	Test AUC	Accuracy	Class 1 Recall
75 features	0.5761	0.5616	0.5596	0.4584
65 features	0.5762	0.5625	0.5597	0.4594
28 features	0.5756	0.5624	0.5594	0.4560

Conclusion: *The 65-feature version provides the most visually aligned signal-to-noise balance, demonstrating clearer agreement with true market direction compared to the other two settings. This aligns with our earlier metrics-based analysis that favored the 65-feature configuration as the best trade-off between complexity and accuracy.*

LightGBM: Data Processing

- Removed 4 features with high missing values
- Leveraged built-in missing value handling (`use_missing=True`)
- Feature selection via importance threshold (38/75 features retained)
- MICE imputation for MLP compatibility

```
Based on median(573.00)Number of features retained after filtering: 38
Examples of reserved features: ['feature_00' 'feature_01' 'feature_02' 'feature_03' 'feature_04'
 'feature_05' 'feature_07' 'feature_08' 'feature_14' 'feature_15'
 'feature_17' 'feature_19' 'feature_20' 'feature_22' 'feature_23'
 'feature_24' 'feature_25' 'feature_28' 'feature_29' 'feature_30'
 'feature_36' 'feature_39' 'feature_47' 'feature_49' 'feature_50'
 'feature_51' 'feature_52' 'feature_53' 'feature_54' 'feature_55'
 'feature_58' 'feature_59' 'feature_60' 'feature_66' 'feature_68'
 'feature_69' 'feature_71' 'feature_72']
```

Data Preprocessing: MICE vs Mean Imputation

MICE (Multiple Imputation by Chained Equations)

- **Principle**: Iterative multivariate imputation
- **Process**:
  1. Build predictive models for each incomplete variable
  2. Iteratively update imputations via chained equations
  3. Generate multiple complete datasets
  4. Pool final results
- **Advantages**:
  - Preserves variable relationships
  - Handles arbitrary missing patterns
  - Provides uncertainty estimates

Mean Imputation

- **Principle**: Simple mean substitution
- **Process**:
  1. Calculate feature means
  2. Fill all missing values with mean

```
# === MICE ===
# sample shape: (3365013, 75)
# using: HistGradientBoostingRegressor
#
# dataset: train
# process chunking: 100%|██████████| 10/10 [02:42<00:00, 16.25s/it]
#
# dataset: val
# process chunking: 100%|██████████| 10/10 [00:29<00:00,  2.95s/it]
#
# dataset: test
# process chunking: 100%|██████████| 10/10 [01:11<00:00,  7.14s/it]
#
# dataset: future
# process chunking: 100%|██████████| 10/10 [00:32<00:00,  3.21s/it]
#
# === time: 28 minute 27.74 sec ===
# save:
# - train: X_train_mice.pkl
# - val: X_val_mice.pkl
# - test: X_test_mice.pkl
# - future: X_future_mice.pkl
```

LightGBM: Model Configuration - Baseline

Key Hyperparameters:

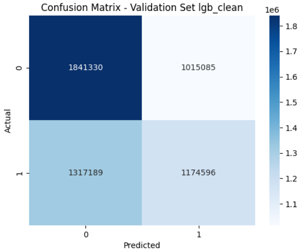
- Tree structure: 511 leaves, unlimited depth
- Regularization: L1/L2=0.05, feature fraction=0.7
- Training: LR=0.02, 1000 rounds
- GPU optimization: 255 max bins

```
=== lgb_clean ===
test AUC: 0.5830 | time: 100.73
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5830    0.6446    0.6123   2856415
           1     0.5364    0.4714    0.5018   2491785

    accuracy                         0.5639   5348200
   macro avg     0.5597    0.5580    0.5570   5348200
weighted avg     0.5613    0.5639    0.5608   5348200

[Val] AUC: 0.5830372749303108
test AUC: 0.5675 | time: 237.07
[TEST] AUC: 0.5674896500636926
```



LightGBM: Alternative Approaches

Method Comparison:

- Method 2: Filtered features
- Method 3: Filtered + MICE + Z-score

```
=== lgb_filtered ===
test AUC: 0.5766 | time: 94.32
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5785    0.6467    0.6107   2856415
           1     0.5318    0.4600    0.4933   2491785

    accuracy                         0.5597   5348200
   macro avg     0.5551    0.5533    0.5520   5348200
weighted avg     0.5567    0.5597    0.5560   5348200

[Val] AUC: 0.5765979914375632
test AUC: 0.5616 | time: 214.41
[TEST] AUC: 0.5615735208682778
```

```
=== lgb_3pre ===
test AUC: 0.5743 | time: 90.90
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5768    0.6482    0.6104   2856415
           1     0.5301    0.4549    0.4896   2491785

    accuracy                         0.5581   5348200
   macro avg     0.5534    0.5515    0.5500   5348200
weighted avg     0.5550    0.5581    0.5541   5348200

[Val] AUC: 0.5743418126637555
test AUC: 0.5593 | time: 207.60
[TEST] AUC: 0.5592606564786361
```

LightGBM: Summary

Original Data Performs Best

- Highest AUC (0.5830) achieved with original data (Method 1)
- Feature engineering & imputation reduced performance:
  - Method 2 (feature selection): AUC $\downarrow$ 0.5732
  - Method 3 (MICE + normalization): AUC $\downarrow$ 0.5706

Superior Minority Class Recall

- Class 1 recall (0.4716) outperforms:
  - Logistic Regression models
  - XGBoost models
- Key advantage: Built-in missing value handling preserves subtle patterns
- Key insight: Leaf-wise growth strategy prioritizes leaf nodes with the greatest split gain and is more sensitive to difficult-to-split samples, usually minority classes.

MLP (Neural Network): Data Processing

- Mandatory MICE imputation
- Feature standardization
- Feature engineering:
  - interaction features created

	Feature	Importance	Selected	Ranking
4	feature_04	2234	True	1
1	feature_01	1833	True	1
15	feature_15	1410	True	1
55	feature_59	1388	True	1
5	feature_05	1349	True	1
48	feature_52	1289	True	1
51	feature_55	1281	True	1
54	feature_58	1227	True	1
50	feature_54	1084	True	1
49	feature_53	1069	True	1

MLP (Neural Network): Baseline Model

Architecture:

- 4 FC layers (256-128-64-1)
- ReLU activations

Training:

- Adam optimizer ($lr = 10^{-3}$)
- BCEWithLogitsLoss
- Batch size=4096, 30 epochs

```
test AUC: 0.5544
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5664    0.6276    0.5955   2856415
           1     0.5128    0.4493    0.4789   2491785

    accuracy                         0.5445   5348200
   macro avg     0.5396    0.5384    0.5372   5348200
weighted avg     0.5414    0.5445    0.5412   5348200

[Val] AUC: 0.5543638987687336
test AUC: 0.5376
[TEST] AUC: 0.5376174109499735
```

MLP (Neural Network): Enhanced Model

Improvements:

- Dropout layers (p=0.3)
- Class weighting (pos_weight)
- Feature selection (top 38 features)

```
Validation AUC: 0.5732
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5828    0.5798    0.5813   2856415
           1     0.5211    0.5242    0.5226   2491785

    accuracy                         0.5539   5348200
   macro avg     0.5520    0.5520    0.5520   5348200
weighted avg     0.5541    0.5539    0.5540   5348200

[Val] AUC: 0.5732464366179105
Test AUC: 0.5562
[TEST] AUC: 0.5562462054863556
```

- We can see that the AUC and recall for Class 1 has increased

MLP (Neural Network): Robust Model

Improvements:

- 45 interaction features
- Residual connections
- Batch normalization
- LeakyReLU ($\alpha = 0.01$)
- AdamW with weight decay ($1e^{-4}$)
- Cyclic learning rate
- Kaiming/Xavier initialization

```
Validation AUC: 0.5706
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5828    0.5573    0.5698   2856415
           1     0.5167    0.5426    0.5293   2491785

    accuracy                         0.5505   5348200
   macro avg     0.5497    0.5500    0.5495   5348200
weighted avg     0.5520    0.5505    0.5509   5348200

Test AUC: 0.5560
```

MLP (Neural Network): Performance Comparison

Model	**Val AUC**	**Test AUC**	**Accuracy**	**Recall (Class 1)**
Baseline	0.5544	0.5376	0.5445	0.4493
Enhanced	0.5732	0.5562	0.5539	0.5242
Robust	0.5706	0.5560	0.5505	**0.5426**

- Feature selection(Enhanced model) improves AUC
- Robust model best for minority class recall
- The LightGBM model and the MLP model each have their own advantages and disadvantages. Next, we will conduct a relevant analysis.

Model Comparison: LightGBM vs MLP

Model	Val AUC	Test AUC	Accuracy	Recall (Class 1)
LightGBM	**0.5830**	**0.5675**	**0.5639**	0.4716
MLP Baseline	0.5544	0.5376	0.5445	0.4493
MLP Enhanced	0.5732	0.5562	0.5539	0.5242
MLP Robust	0.5706	0.5560	0.5505	**0.5426**

Despite comparable accuracy across models - reflecting inherent data noise and metric limitations - this performance dichotomy highlights LightGBM's advantage in overall ranking capability versus MLP's strength in minority class identification, stemming from their fundamental architectural differences in handling data relationships and error tradeoffs

Prediction Visualization

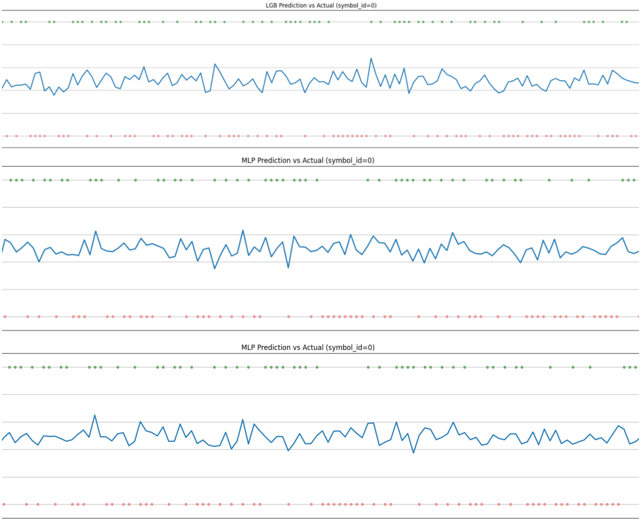


Figure: LightGBM baseline vs MLP Enhanced vs MLP Robust

Conclusion

Key Findings:

- LightGBM: Best overall performance (AUC 0.5830)
- Robust MLP: Best minority recall (0.5426)
- Comparable accuracy across models (0.55)

Limitations & Future Work:

- Dataset challenges: Noise, weak signals
- Hybrid approaches recommended
- Advanced feature engineering needed
- TabNet/TabTransformer exploration

Conclusion

Overall, the XGBoost model with 65 selected features and LightGBM models offered the best balance of interpretability, complexity, and efficiency. For scenarios prioritizing minority class recall, Robust MLP is preferable. Nevertheless, the relatively limited gains across all models indicate a need for further exploration into model design and feature representation to fully realize the predictive potential of financial data.

Thank You