OUT OF STOCK CLASSIFICATION PREDICTION

Jae Kang, Xin Wang, Harmandeep Teja, Ayman Shahriar

ABSTRACT

Skip, a Canadian delivery app, has posed a hypothetical business problem where deliveries fail due to out-of-stock (OOS) items at partner restaurants and stores, as Skip lacks access to their inventory data. To address this issue, we developed machine learning (ML) models that predict whether an incoming order is in-stock or OOS based on historical order data.

Our approach involved exploratory data analysis (EDA) and feature engineering, followed by the implementation of two models: a gradient-boosted tree using Scikit-learn and a fully-connected neural network (FNN) using PyTorch. The dataset was split 80/20 for training and testing, and performance was evaluated using relevant metrics. Preprocessing techniques included handling class imbalance through SMOTE. Results show that the gradient-boosted tree delivered consistent performance across different feature sets, with a robust detection rate of OOS events. In contrast, the FNN, particularly when trained using all features, achieved higher sensitivity by minimizing false negatives—albeit at the expense of an increased number of false positives. This trade-off aligns well with our goal of prioritizing the detection of OOS items. Overall, our findings provide valuable insights into the effectiveness of different machine learning approaches in predicting OOS situations and lay the groundwork for future enhancements, such as more diverse data and real-world validation to further enhance model effectiveness.

1. INTRODUCTION

This mock project is provided by our industry partner Skip, a Canadian company that offers a delivery app, in which a customer can order food and groceries from Skip's partner restaurants and stores. Skip has posed a hypothetical business problem with regards to the out-of-stock (OOS) issue, where a delivery of an incoming order is attempted but fails because the deliverer finds out that the ordered items are OOS at the pick-up restaurant/store. This issue happens because Skip does not have access to the inventory data of their partner restaurants and grocery stores.

To address this issue, we developed ML models that predict whether an incoming order is in-stock or OOS based on historical orders. The given data includes 236,750 orders and each order includes one or more of the same item. Each order data includes order details and whether the order was successful or not (due to OOS). To address Skip's OOS issue, we decided to create a machine learning (ML) model that predicts whether an incoming order is in-stock or OOS.

2. RELATED WORK

Past research addressing on-shelf availability and out-of-stock (OOS) issues has primarily focused on careful feature selection and the use of various machine learning models to enhance retail performance. For instance, researchers have examined time-based patterns—such as the day of the week—alongside sales and inventory data to capture the dynamics affecting product availability [1]. Studies have applied models like Decision Trees to identify influential features, while others have employed probabilistic approaches such as Naive Bayes and Hidden Markov Models to model temporal dependencies and uncertainty in retail environments [2][3]. Chuang et al. explored how external audits and on-shelf availability can impact retail outcomes, underlining the importance of robust predictive models [1]. Similarly, Montoya and Gonzalez demonstrated the effectiveness of a Hidden Markov Model in detecting on-shelf OOS situations using point-of-sale data [2], and Papakiriakopoulos and Doukidis investigated classification performance for decision-making regarding missing products [3].

The key distinction between our project and these earlier studies is that we work exclusively with historical order data obtained from Skip, a Canadian delivery app. Unlike previous research that leverages rich datasets including inventory and sales records, our approach contends with the challenge of predicting OOS events without direct access to such data. This limitation necessitates a more nuanced strategy for extracting latent indicators of product availability solely from order information, setting our work apart and highlighting the need for innovative solutions in data-constrained environments.

3. MATERIALS AND METHODS

All data processing and model development was conducted using Google BigQuery, a fully managed data warehouse designed for handling large-scale datasets efficiently. A key advantage of BigQuery is its ability to store and manage SQL tables, queries and python notebooks, making it possible to train and evaluate models without requiring external computing resources.

The dataset used for this study consists of historical order data provided by Skip, containing information on past delivery attempts and relevant order attributes (Table 1). Each delivery has a binary label associated with it that describes whether the delivery was successful or failed due to the item being out of stock. The data includes 236,750 orders and each order includes one or more of the same item.

Feature Name	SQL Data Type
unique_id	STRING
order_item_id	STRING
createdTime	TIMESTAMP
adjustmentId	STRING
restaurant_name	STRING
order_item_quantity	INT64
order_number	STRING
restaurant_id	STRING
order_item_name	STRING
order_item_price_each	FLOAT64
orderId	STRING
order_item_price_sum	FLOAT64
total	FLOAT64
subtotal	FLOAT64
scenario (whether the order was out of stock or not)	STRING

Table 1. Historical Order Data Features and Data Types

This raw data undergoes multiple preprocessing steps, including exploratory data analysis (EDA) to understand its structure, data inspection to identify missing or inconsistent values, data filtering to remove noise or irrelevant entries, and feature engineering to construct meaningful predictors for the model. This results in a transformed dataset that serves as the foundation for model training and evaluation. To ensure a robust evaluation of the model's performance, the transformed dataset is divided into two distinct subsets: 80% is allocated for the development set, used for training and hyperparameter tuning, and 20% is reserved as the testing set to evaluate the final model's generalization to unseen data.

We will use the data to create two types of models—a fully connected neural network [4] using PyTorch and a gradient boosted tree model [5] using scikit-learn. We scaled the data using standard scaler [6] before training the neural network model, and utilized ReLU [7] as the activation function and Adam [8] for the optimization algorithm. Once the models are trained, we evaluated the comparative model performance by generating the confusion matrix of each model using the test set.

Feature engineering was essential to extract and construct meaningful variables from our raw data, ensuring our models could capture the key patterns and trends needed for accurate OOS predictions. We first exhaustively generated as many meaningful features from the raw data as possible. To efficiently create the new features (some of which involve complex processing of the raw data), we used SQL queries to carry out most of the feature engineering.

After creating a wide range of new features, we trained a boosted tree model using all the features, and then generated the list of feature importances of that model (the concept of feature importance is discussed from the early works of decision trees [9][10][11]). Following that, we built boosted tree and neural network models using all the features, the top 10 features, and the top 5 features. This systematic evaluation helped us pinpoint which features were most valuable, ensuring that our final model was both efficient and effective. We also decided to restrict any time-related features to have hourly granularity, as including minutes and seconds might be too fine grained for our use case.

Out of the 236,750 orders in our dataset, 5,634 of them are labelled as "out of stock" while the other 230,936 are labelled as "delivered", creating a significant class imbalance that can adversely affect model performance. To address this, we employ SMOTE [12] from the imblearn.over_sampling module—a data augmentation technique that generates synthetic examples for the minority class. By using SMOTE, we artificially boost the number of "out of stock" datapoints to reach a 1:4 ratio with the "delivered" class, thereby creating a more balanced dataset. This balanced representation ensures that the model learns effectively from both classes, reducing bias toward the majority class and improving overall predictive performance.

4. RESULTS AND DISCUSSION

4.1. Key EDA Findings

- The data table contains 13 columns (including unique_id) and 236,570 rows.
- The data covers from January 1, 2025, to November 1, 2025.
- 230,936 of the orders are delivered successfully and 5,634 of them are OOS.
- The table has 11 unique restaurant IDs and 7 unique restaurant names, meaning 4 restaurants have the same names.
- Each restaurant has both delivered and OOS orders.
- The table has 20 unique items; each item has both delivered and OOS orders.
- Some items are sold at more than one restaurant.
- Each order contains only one type of item, but the quantity of that item can be more than one.

Additional EDA was carried out but not mentioned in the report as their results were deemed to be irrelevant to model development and evaluation. For the full EDA carried out in this study, please refer to the exploratory data analysis notebook that is included in the repository.

4.2. Feature Engineering

We have generated new features that belong to three categories:

4.2.1. Time-Based Features

- Hour of day
- Weekday or weekend
- Quarter of year
- AM or PM
- Day of month

4.2.2. Features for Each Item at a Restaurant/Store

For each item at a specific restaurant or store, we generate new features based on the order count, OOS events, and restocking. We count things over different time periods: 1 hour, 3 hours, 6 hours, one day, a week, and a month. These features include:

- Number of orders in the past 1 hour, 3 hours, 6 hours, 1 day, 3 days, 1 week
- Quantity ordered in the past 1 hour, 3 hours, 6 hours, 1 day, 3 days, 1 week
- Time elapsed since last order (could be either delivered or out of stock)
- Time elapsed since last delivered order
- Time elapsed since last out of stock order
- Number of out of stock orders in the past 1 hour, 3 hours, 6 hours, 1 day, 3 days, 1 week
- Number of delivered orders in the past 1 hour, 3 hours, 6 hours, 1 day, 3 days, 1 week
- Time elapsed since last restock

4.2.3. Features for Each Restaurant/Store

These features are similar the features for each item at a specific store, but instead of considering a single item at a store, these features take into consideration all items at a single store:

- Number of orders in the past 1 hour, 3 hours, 6 hours, 1 day, 3 days, 1 week
- Quantity ordered in the past 1 hour, 3 hours, 6 hours, 1 day, 3 days, 1 week
- Time elapsed since last order (could be either delivered or out of stock)
- Time elapsed since last delivered order
- Time elapsed since last out of stock order
- Number of out of stock orders in the past 1 hour, 3 hours, 6 hours, 1 day, 3 days, 1 week
- Number of delivered orders in the past 1 hour, 3 hours, 6 hours, 1 day, 3 days, 1 week
- Time elapsed since last restock

Feature	**Feature Importance Score**
Time elapsed since last out of stock order (specific to the item)	0.985684
Time elapsed since last order (specific to the item)	0.004843
Time elapsed since last delivered order (specific to the item)	0.002993
Time elapsed since last restock (specific to the restaurant)	0.001263
Quantity ordered in the past 3 days (specific to the restaurant)	0.000993
Time elapsed since last restock (specific to the restaurant)	0.000928
Number of orders in the past 3 days (specific to the restaurant)	0.000440
Number of orders in the past 1 day (specific to the restaurant)	0.000438
Number of out of stock orders in the past 1 week (specific to the restaurant)	0.000348
Quantity ordered in the past 1 day (specific to the restaurant)	0.000341

Table 2. Top 10 Features Ordered By Feature Importance

After training the boosted tree model using all features (including the engineered features), we generated how important each feature is to the model using the .feature_importances_ attribute (table 2). The top two features have a combined importance of more than 99%. This is a strong indicator that the model will perform well using a very small number of features. Most of the top features are specific to each item, while the lower-ranked features are specific to the restaurant. This intuitively makes sense, as information that is specific to the single ordered item will be more useful to the model than information that is aggregated over all items in the restaurant.

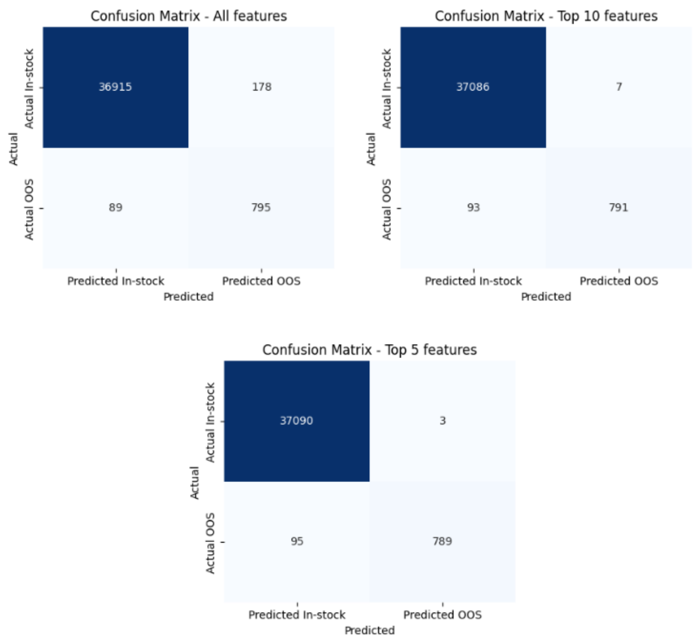

Fig. 1. Confusion Matrices of the Fully Connected Neural Network Models

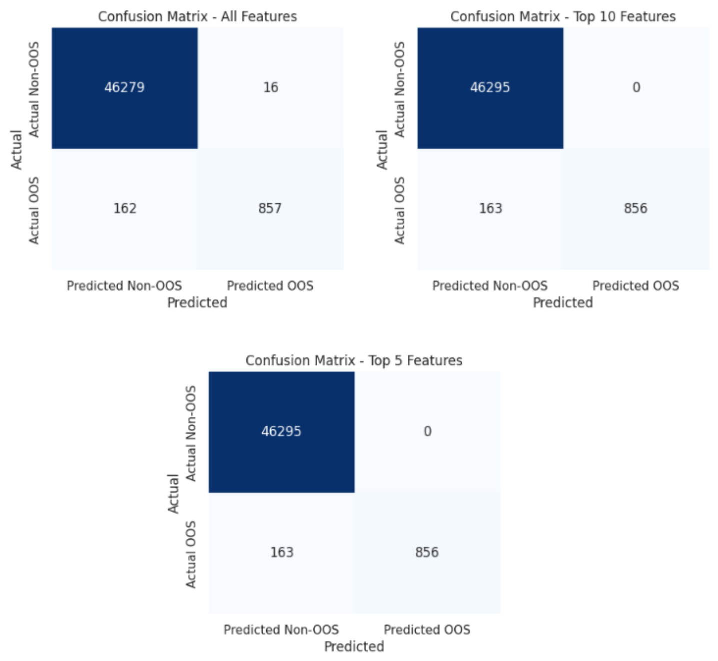

Fig. 2. Confusion Matrices of the Boosted Tree Models

4.3. Analysis of Neural Network Model Results

The results (Figure 1) indicate a clear trade-off between sensitivity and specificity as the number of features is reduced. The model trained with all features has a higher number of false positives among in-stock orders (178), but has the smallest number of false negatives (89). In contrast, the top 10 and top 5 features models drastically reduce false positives (7 and 3, respectively), which improves specificity; however, this comes with a slight increase in false negatives. In other words, while reducing the number of features leads to fewer misclassifications of in-stock items as OOS, it also results in missing a few more out-of-stock cases.

Although the all-features model produces more false positives, it minimizes false negatives, ensuring that fewer OOS items are missed. In contrast, the models using 10 or 5 features tend to have fewer false positives but miss more OOS cases. Since our goal is to reliably flag out-of-stock items, the increased sensitivity of the all-features model aligns best with our priorities.

4.4. Analysis of Gradient Boosted Tree Results

All three gradient boosted tree models demonstrate similar overall performance. The model trained on all features registers 16 false positives and 162 false negatives, while both the top 10 and top 5 features models show no false positives but have one additional false negative (163). This indicates that reducing the feature set slightly improves specificity—with no false positives—but comes at the cost of a marginal decrease in sensitivity. Given these results, while the all-features model may offer a tiny edge in capturing OOS cases, the simpler top 10 or top 5 features models provide nearly equivalent performance with potentially greater interpretability and efficiency.

4.5. Comparison of the two models

Given that our objective is to prioritize the detection of out-of-stock items—even if it means accepting a higher rate of false positives—the neural network model trained using all features is very promising. In our neural network experiments, this all-features model exhibited the lowest number of false negatives compared to its 5- and 10-feature counterparts, meaning it missed fewer OOS cases. On the other hand, the gradient boosted tree classifiers showed very consistent performance across all three feature sets (all, 10, and 5 features) with roughly 84% recall for OOS items. Their false negatives and false positives were nearly identical regardless of the feature subset used.

Thus, if maximum sensitivity to OOS items is the primary business requirement, deploying the all-features neural network model would be ideal despite its higher false positive rate. However, if we value consistency, interpretability, and potentially lower computational overhead, one of the gradient boosted tree models (any of the three given their near-identical performance) would serve as a robust alternative.

5. STRENGTHS AND LIMITATIONS OF THE PROPOSED METHODOLOGY

5.1. Strengths

Diverse Modeling: By implementing both a fully connected neural network (PyTorch) and a gradient boosted tree model (scikit-learn), the approach allows for robust comparative analysis and selection based on performance.

Addressing Class Imbalance: The use of SMOTE to synthetically balance the minority class ensures the models remain sensitive to out-of-stock events, which is crucial for business objectives.

Robust Evaluation: An 80/20 train-test split, along with multiple performance metrics, safeguards against data leakage and provides a reliable measure of generalization.

5.2. Limitations

Limited Data Scope: Each order includes only one or more of the same item, a constraint that may oversimplify real-world ordering behavior and limit the model's applicability to more complex scenarios.

Synthetic Data Dependency: SMOTE-generated examples may not fully capture real-world complexities, risking overfitting or misrepresentation of actual out-of-stock scenarios.

6. CONCLUSION

Our study demonstrates that even with limited data—restricted to historical order details and single-item orders—a robust machine learning approach can effectively predict out-of-stock situations. By leveraging comprehensive EDA, systematic feature engineering, and addressing class imbalance through SMOTE, we built two distinct models using both neural network and gradient boosted tree architectures. The comparative analysis revealed that while the fully connected neural network trained with the entire feature set achieves higher sensitivity to OOS events, the gradient boosted tree offers consistent performance with greater interpretability and lower computational overhead. Despite the inherent limitations, including the simplified order structure and reliance on synthetic minority augmentation, our findings offer valuable insights for maximizing revenue in a hypothetical business scenario. Future work should incorporate more diverse data and real-world validation to further enhance model effectiveness.

14. REFERENCES

[1]: Chuang, H. H., Oliva, R., & Liu, S. (2016). On-shelf availability, retail performance, and external audits: A field experiment. Product and Operations Management.

[2]: Montoya, R., & Gonzalez, C. (2019). A Hidden Markov Model to Detect On-shelf Out-of-stocks Using Point of Sales Data. Manufacturing & Service Operations Management 21.

[3]: Papakiriakopoulos, D. A., & Doukidis, G. I. (2011). Classification performance for making decisions about products missing from the shelf. Advances in Decision Sciences.

[4]: Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning representations by back-propagating errors. Nature, 323, 533-536.

[5]: Friedman, J. (2001) Greedy Function Approximation: A Gradient Boosting Machine, The Annals of Statistics, Vol. 29, No. 5.

[6]: Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). “Scikit-learn: Machine Learning in Python.” Journal of Machine Learning Research, 12, 2825–2830.

[7]: Vinod Nair, Geoffrey E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines" 2010.

[8]: Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[9]: Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). Classification and regression trees. CRC Press.

[10]: Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

[11]: Breiman, L., & Friedman, J. H. (1985). Estimating Optimal Transformations for Multiple Regression and Correlation. Journal of the American Statistical Association, 80(391), 580–598. https://doi.org/10.1080/01621459.1985.10478157

[12]: Chawla, N. V., Bowyer, K. W., Hall, L. O., Kegelmeyer, W. P.. (2002). “SMOTE: synthetic minority over-sampling technique,” Journal of artificial intelligence research, 321-357.