Capstone Project **Tran Phan Quoc Dat**
Data Scientist Nano Degree **December 15, 2024**

1. Project Definition

Project Overview

This project focuses on forecasting the sales of various products sold by Rohlik Group, one of the leading online grocery delivery companies. The problem domain revolves around accurately predicting sales quantities for products in different warehouses, leveraging historical sales data, inventory characteristics, and calendar information, including holidays and other temporal factors. The project originated from the Kaggle competition, "Rohlik Sales Forecasting Challenge V2," and aims to solve a critical business need: improving stock management and operational efficiency.

The dataset provided contains five files:

- sales_train.parquet: Historical daily sales data for each product.
- calendar.csv: Information about holidays and special events.
- inventory.csv: Metadata about product categories and their availability.
- sales_test.csv: Test data for generating sales forecasts for evaluation.
- test_weights.csv: Weights corresponding to the importance of individual products for calculating the competition score.

The core objective of this project is to build a machine learning model capable of predicting daily sales for the test dataset, enabling effective inventory management and minimizing stock-outs or overstock scenarios.

Problem Statement

The key challenge is to accurately forecast the daily sales quantities for various products in multiple warehouses based on their historical sales data, inventory attributes, and calendar features. This prediction task is critical for optimizing inventory planning and logistics operations. Misestimations can lead to significant business inefficiencies, such as excess inventory costs or missed sales opportunities due to stock-outs.

The solution must generalize well to unseen data and address potential issues such as seasonality, data sparsity, and fluctuating sales patterns for specific products.

Metrics

The metric used to evaluate model performance is **Weighted Mean Absolute Percentage Error (WMAPE)**, the same as specified in the Kaggle competition. WMAPE is calculated as:

$$\text{WMAPE} = \frac{\sum_{i=1}^{n} w_i \cdot |y_i - \hat{y}_i|}{\sum_{i=1}^{n} w_i \cdot y_i}$$

Where:

- y_i: Actual sales for product i,
- $\hat{y}_i$: Predicted sales for product i,
- w_i: Weight assigned to product i,
- n: Total number of products.

WMAPE was chosen because it effectively captures the relative error weighted by product importance. This metric ensures that errors in predicting high-priority products (those with larger weights) are penalized more heavily, aligning the evaluation process with real-world business needs.

2. Analysis

Data Exploration

The data is described based on information from Kaggle.

sales_train.csv and **sales_test.csv**

- unique_id - unique id for inventory
- date - date
- warehouse - warehouse name
- total_orders - historical orders for selected Rohlik warehouse known also for test set as a part of this challenge
- sales - Target value, sales volume (either in pcs or kg) adjusted by availability. The sales with lower availability than 1 are already adjusted to full potential sales by Rohlik internal logic. There might be missing dates both in train and test for a given inventory due to various reasons. This column is missing in test.csv as it is target variable.
- sell_price_main - sell price
- availability - proportion of the day that the inventory was available to customers. The inventory doesn't need to be available at all times. A value of 1 means it was available for the entire day. This column is missing in test.csv as it is not known at the moment of making the prediction.
- type_0_discount, type_1_discount, … - Rohlik is running different types of promo sale actions, these show the percentage of the original price during promo ((original price - current_price) / original_price). Multiple discounts with different type can be run at the same time, but always the highest possible discount among these types is used for sales. Negative discount value should be interpreted as no discount.

inventory.csv

- unique_id - inventory id for a single keeping unit
- product_unique_id - product id, inventory in each warehouse has the same product unique id (same products across all warehouses has the same product id, but different unique id)
- name - inventory id for a single keeping unit
- L1_category_name, L2_category_name, … - name of the internal category, the higher the number, the more granular information is present
- warehouse - warehouse name

calendar.csv

- warehouse - warehouse name
- date - date
- holiday_name - name of public holiday if any
- holiday - 0/1 indicating the presence of holidays
- shops_closed - public holiday with most of the shops or large part of shops closed
- winter_school_holidays - winter school holidays
- school_holidays - school holidays

test_weights.csv

- unique_id - inventory id for a single keeping unit
- weight - weight used for final metric computation

Sample data:

```
train_sales_df.head()
```

	unique_id	date	warehouse	total_orders	sales	sell_price_main	availability	type_0_discount	type_1_discount	type_2_discount	type_3_discount	type_4_discount	type_5_discount	type_6_discount
0	4845	2024-03-10	Budapest_1	6436.0	16.34	646.26	1.00	0.00000	0.0	0.0	0.0	0.15312	0.0	0.0
1	4845	2021-05-25	Budapest_1	4663.0	12.63	455.96	1.00	0.00000	0.0	0.0	0.0	0.15025	0.0	0.0
2	4845	2021-12-20	Budapest_1	6507.0	34.55	455.96	1.00	0.00000	0.0	0.0	0.0	0.15025	0.0	0.0
3	4845	2023-04-29	Budapest_1	5463.0	34.52	646.26	0.96	0.20024	0.0	0.0	0.0	0.15312	0.0	0.0
4	4845	2022-04-01	Budapest_1	5997.0	35.92	486.41	1.00	0.00000	0.0	0.0	0.0	0.15649	0.0	0.0

```
calendar_df.head()
```

	date	holiday_name	holiday	shops_closed	winter_school_holidays	school_holidays	warehouse
0	2022-03-16	NaN	0	0	0	0	Frankfurt_1
1	2020-03-22	NaN	0	0	0	0	Frankfurt_1
2	2018-02-07	NaN	0	0	0	0	Frankfurt_1
3	2018-08-10	NaN	0	0	0	0	Frankfurt_1
4	2017-10-26	NaN	0	0	0	0	Prague_2

```
inventory_df.head()
```

	unique_id	product_unique_id	name	L1_category_name_en	L2_category_name_en	L3_category_name_en	L4_category_name_en	warehouse
0	5255	2583	Pastry_196	Bakery	Bakery_L2_14	Bakery_L3_26	Bakery_L4_1	Prague_3
1	4948	2426	Herb_19	Fruit and vegetable	Fruit and vegetable_L2_30	Fruit and vegetable_L3_86	Fruit and vegetable_L4_1	Prague_3
2	2146	1079	Beet_2	Fruit and vegetable	Fruit and vegetable_L2_3	Fruit and vegetable_L3_65	Fruit and vegetable_L4_34	Prague_1
3	501	260	Chicken_13	Meat and fish	Meat and fish_L2_13	Meat and fish_L3_27	Meat and fish_L4_5	Prague_1
4	4461	2197	Chicory_1	Fruit and vegetable	Fruit and vegetable_L2_17	Fruit and vegetable_L3_33	Fruit and vegetable_L4_1	Frankfurt_1

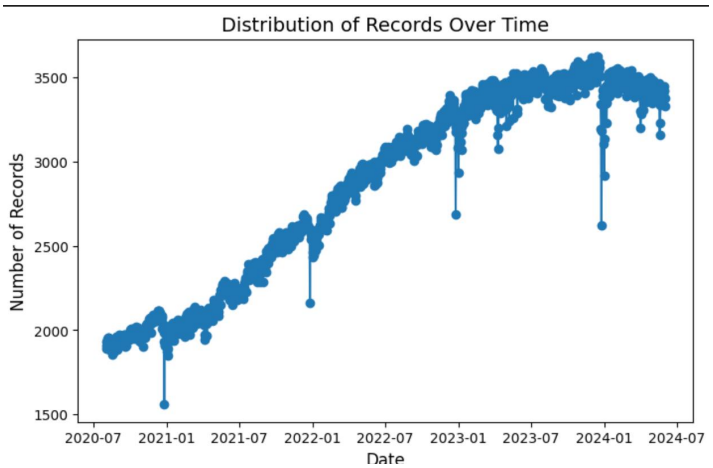
i. In the training data, the number of NaN values is minimal, as shown in the figure below

```
train_sales_df.isna().mean()
```

```
unique_id          0.000000
date               0.000000
warehouse          0.000000
total_orders       0.000013
sales              0.000013
sell_price_main    0.000000
availability       0.000000
type_0_discount    0.000000
type_1_discount    0.000000
type_2_discount    0.000000
type_3_discount    0.000000
type_4_discount    0.000000
type_5_discount    0.000000
type_6_discount    0.000000
dtype: float64
```

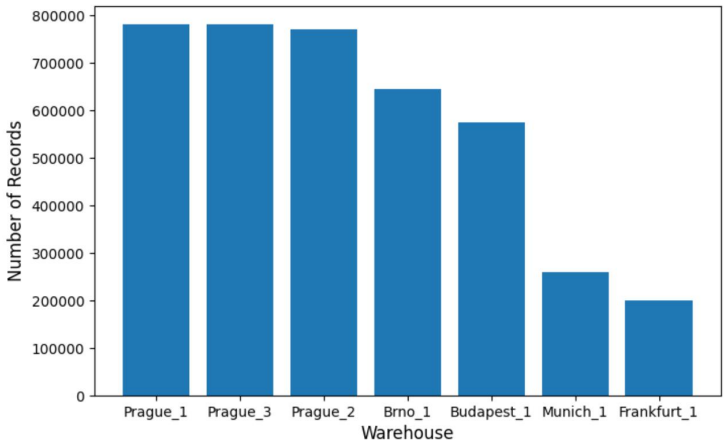
So we need remove records have missing values

ii. The distribution of records over time, checking when the data was most densely collected



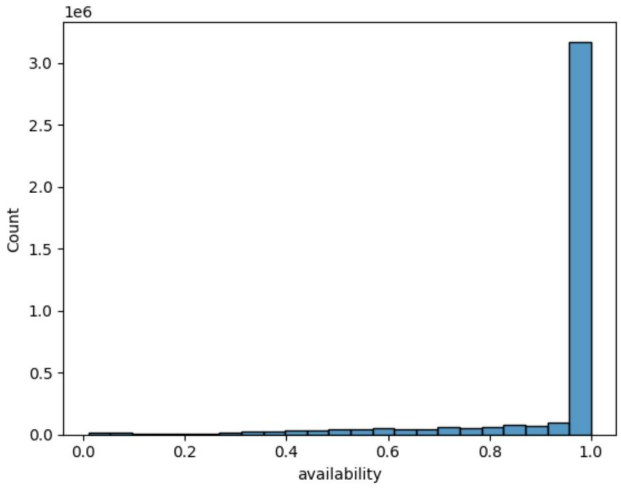
We can see that the data is more concentrated in the recent time period but not significantly difference from previous years

iii. Warehouses distribution in data



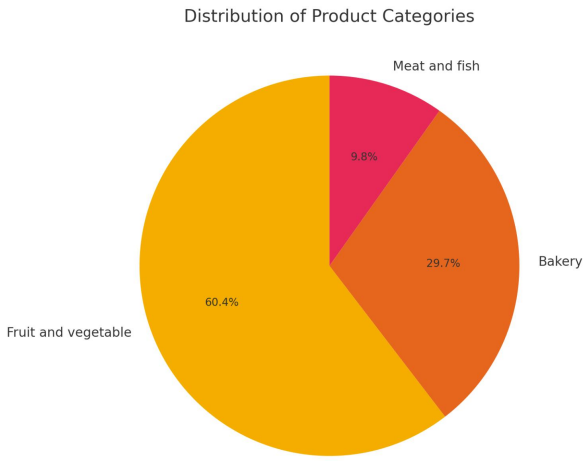
The warehouses have relatively similar quntities, however, Munich_1 and Frankfurt_1 have fewer records compared to the others

iv. The availability column is present in the training data but not in the test data, so we will examine its distribution



Most of the values in this column are 1, so it can be removed from the training data without significantly affecting the results

v. The distribution of product categories in the data

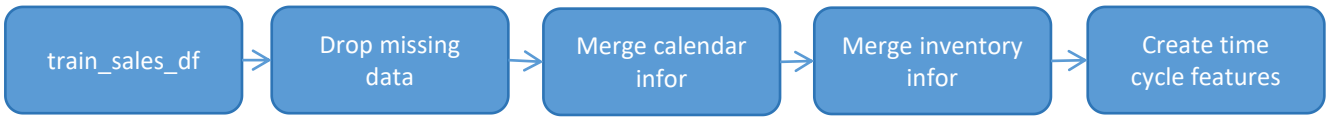


The number of records for sales data of Fruit and Vegetable accounts for the majority, followed by Bakery

3. Methodology

Data Processing

The data processing steps are shown in the diagram below



First, the sales data will be processed to handle null values and then merged with calendar and inventory information to extract details about holidays and product information.

Next, additional features related to time cyclicality will be created. Why are these features needed?
=> The method used here is tree-based, which is not specifically designed to handle time-related data. Therefore, it is necessary to create features that capture cyclicality, such as month, quarter, sin(month), cos(month), etc. These features will help the model better learn the cyclic patterns in the data.

```python
def data_processing(sales_df, calendar_df, inventory_df):
    sales_df['date'] = pd.to_datetime(sales_df['date'])
    # Mapping holidays infor
    df = pd.merge(sales_df, calendar_df, on=['date', 'warehouse'], how='left')

    # Mapping product_infor
    df = pd.merge(df, inventory_df, on=['unique_id', 'warehouse'], how='left')

    # Convert NaN values in holiday_name to empty string
    df['holiday_name'] = df['holiday_name'].fillna('')

    # Create year, month, date cols
    df['year'] = df['date'].dt.year
    df['quater'] = df['date'].dt.quarter
    df['month'] = df['date'].dt.month
    df['day'] = df['date'].dt.day
    df['month_name'] = df['date'].dt.month_name()
    df['day_of_week'] = df['date'].dt.day_name()
    df['week'] = df['date'].dt.isocalendar().week
    df['year_sin']  = np.sin(2*np.pi*df['year'])
    df['year_cos'] = np.cos(2*np.pi*df['year'])

    df['month_sin']  = np.sin(2*np.pi*df['month']/12)
    df['month_cos'] = np.cos(2*np.pi*df['month']/12)

    df['day_sin']  = np.sin(2*np.pi*df['day']/31)
    df['day_cos'] = np.cos(2*np.pi*df['day']/31)
    df['group'] = (df['year'] - 2020)*48 + df['month']*4 + df['day']//7

    # Drop some cols not use for prediction
    if 'availability' in df.columns:
        df = df.drop(columns=['date', 'availability'], axis=1)
    else:
        df = df.drop(columns=['date'], axis=1)


    # Convert category datatype
    cate_cols = ['warehouse', 'name', 'holiday_name', 'L1_category_name_en', 'L2_category_name_en', 'L3_category_name_en', 'L4_category_name_en', 'month_name', 'day_of_week']
    df[cate_cols] = df[cate_cols].astype('category')

    return df
```

Implementation

To tackle the forecasting problem, I used the LightGBM framework, which is well-suited for tabular and time-series data. The core implementation involved training the model using time-based cross-validation with the following setup:

i. Cross-validation setup:

- The entire dataset was divided into **5 folds**, where each fold represented a time-split.
- For each fold, the last 2 weeks of data were used as the validation set, while the remaining earlier data was used as the training set.

- After each fold, the training and validation periods were shifted backward by 2 weeks to ensure no data leakage.

Explain for Cross Validation for Timeseries

ii. Training configuration:

- Hyperparameters: The LightGBM model was configured with the following parameters:
  - objective: Regression.
  - metric: RMSE (Root Mean Squared Error).
  - boosting_type: GBDT (Gradient Boosting Decision Trees).
  - learning_rate: 0.2
  - num_leaves: 31
  - min_data_in_leaf: 25
- Custom evaluation metric: Weighted Mean Absolute Percentage Error (WMAPE) was implemented as a custom evaluation metric to align with the competition scoring criteria. This metric calculates the prediction error while accounting for the importance of different products (weights).

iii. Training loop:

- For each epoch, the model was updated with the training data using the update() function.
- Predictions were made on the training and validation sets at every 10 epochs to compute the WMAPE for both sets.
- **Early stopping**: If the validation loss did not improve for 3 consecutive evaluations, the training was halted early, and the best model iteration was saved.

iv. Model saving:

- After each fold, the trained model was saved using model.save_model() with the best iteration determined by early stopping.

Refinement

To ensure robust model performance, the following refinement techniques were applied:

Cross-validation:

- The 5-fold time-based cross-validation setup ensured that the model was evaluated on multiple unseen validation sets, mimicking the real-world scenario where future data is predicted using past information.
- This method helped identify the most generalizable hyperparameters and reduced overfitting.

Early stopping:

- Early stopping prevented the model from overfitting to the training data, as training was halted if the validation WMAPE did not improve for 3 consecutive evaluations.

Ensembling:

- The final predictions were generated by averaging the outputs of the 5 models trained on different folds. This simple ensembling approach further improved prediction stability and reduced variance.

Weight integration:

- To ensure accurate WMAPE calculation, weights for each product were retrieved and applied during validation. This ensured that model evaluation aligned perfectly with the competition's scoring system.

4. Results

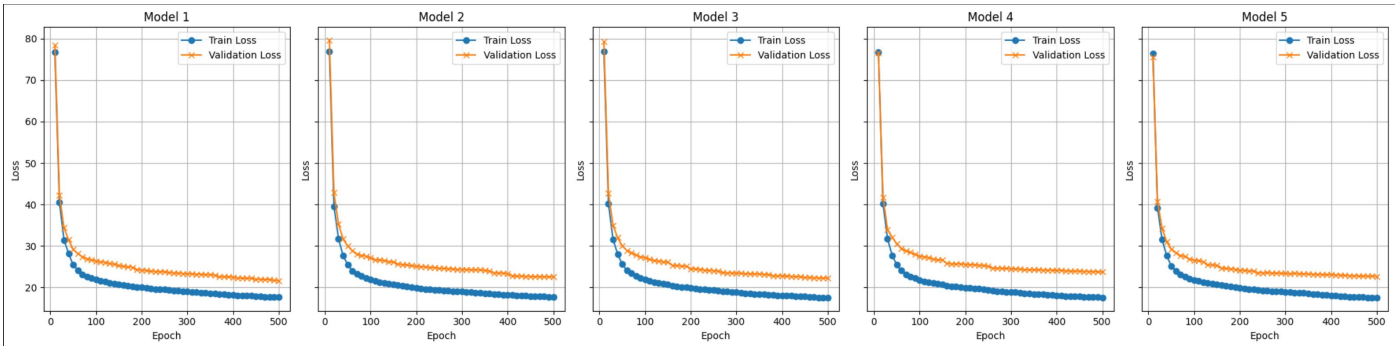
Model Evaluation and Validation

In this problem, there are various approaches, such as specialized time series models like Prophet or ARIMA. However, given the need to predict for many products, the tree-based model approach is more convenient, as it allows a single model to be trained and used for predictions across all products.

The experiments I conducted during the project included:

- **Parameters:**
  - num_leaves: 31,
  - min_data_in_leaf: 25,
  - learning_rate: 0.2
  - These parameters were also adjusted and tested with various other values.
- **Data splitting:**
  - Used random train/validation splitting with the train_test_split function from scikit-learn.
  - Split train and validation data based on time instead of random splits.
- Implemented **cross-validation** and **ensembling** techniques.

The training results with 5-fold cross-validation are as follows:



The loss function in the graph represents the evaluation metric, which is the WMAE score.
Since the validation set only spans 2 weeks, the variations in the training set across different folds are minimal. As a result, the training outcomes of the 5 models are very similar.

Justification

The final result was evaluated by submitting the test set predictions to Kaggle, where the WMAE score was **24.54469** using the cross-validation approach and averaging the predictions from 5 models for ensembling.

Additionally, I submitted other versions to Kaggle for comparison. Below is the comparison table:

No.	Model Details/Training Method	WMAE
1	Used random train/validation split with scikit-learn's `train_test_split` function at a ratio of 0.8/0.2.	**69.62**
	- Parameters: `num_leaves`: 31, `min_data_in_leaf`: 25, `learning_rate`: 0.2	
2	Split train/validation by time instead of random (used the last 2 weeks for validation).	**23.93**
	- Parameters: `num_leaves`: 31, `min_data_in_leaf`: 25, `learning_rate`: 0.2	
3	Split train/validation by time instead of random (used the last 2 weeks for validation).	**26.56**
	- Parameters: `num_leaves`: 45, `min_data_in_leaf`: 30, `learning_rate`: 0.15	
4	Performed cross-validation and ensembling.	**24.54**
	- Parameters: `num_leaves`: 31, `min_data_in_leaf`: 25, `learning_rate`: 0.2	

It can be observed that using only one fold for prediction yields the best results, even better than the ensemble of 5 models from 5 folds.
The reason for this could be that the other models were trained with training data shifted by 2 weeks compared to the prediction period. As a result, the time gap between training and testing is larger, causing the performance of those models to degrade. Therefore, using an average when ensembling makes the overall result worse compared to using only fold 1.

Additionally, in setting (1), the results are very poor. In this case, the evaluation scores on the training and validation sets were both below 18, but the WMAE on the test set was **69.62**. This is because the model was overfitted. Since the validation set was chosen randomly, the model might have had access to information from both before and after the samples in the validation set, leading to overfitting.

In setting (3), the model also experienced overfitting but for a different reason: the model became more complex, which led to overfitting. To address this, I reduced the number of num_leaves and increased min_data_in_leaf.

5. Conclusion

Reflection

In this project, I conducted an analysis of the problem requirements and data characteristics. This is a complex problem because the data is highly influenced by external factors that are difficult to predict, and the target values also depend on previous values, similar to a time series. This presented a significant challenge: designing features that are suitable for the problem. For example, to capture seasonality, it was necessary to create cyclic time features such as quarter, sin(month), cos(month), etc.

From an implementation perspective, I experimented with possible solutions such as cross-validation and ensembling. While the results were not yet optimal, I believe the ideas behind these approaches can serve as a foundation for improving model accuracy in the future.

Improvement

The following solutions could be applied to improve the model:

1. **Parameter Tuning**: Add more hyperparameters and use grid search for parameter tuning instead of manually experimenting as done currently.
2. **Feature Engineering for Products**: Currently, the product information is grouped into three main categories. Additional features could be introduced to specify more detailed types of products. For example, instead of having just "meat" as a feature, we could introduce attributes like "pork," "beef," etc.
3. **Trend and Seasonal Attributes**: Add features related to trends and seasonality to make it easier for the model to learn. These attributes could be computed using time series decomposition methods such as STL.
4. **Exploring Other Models or Frameworks**: Experiment with other models or frameworks such as XGBoost or CatBoost.
5. **Encoding or Reprocessing Categorical Features**: Improve how categorical data is encoded or processed to enhance model performance.
6. **Data Normalization**: Normalize the data if using models that require normalized inputs.

These improvements could help address current model limitations and enhance its ability to make more accurate predictions.