# Instacart Market Analysis

Group 10:

Tiantian Zhang, Shengshi Yuan, Fan Zhang, Xinyi Tan, Siyu Shen

# Introduction

- Instacart is a growing e-commerce company that provides grocery delivery and pick-up services.
- With the massive amount of customer data, the goal of this project is to **predict which products the users will reorder next**, so that the company can further boost revenue.
- In this project, we examined different predictive models and optimize the performance of our final model.


Photo Source: https://www.supermarketnews.com/online-retail/instacart-sees-2020-year-grocery-pickup

instacart

# Dataset

- The dataset from the Kaggle competition [1] is a relational set of files describing customers’ orders over time.

| Order_products_prior.csv | Order_products_train.csv | Orders.csv | Products.csv | Aisles.csv | Departments.csv |
|---|---|---|---|---|---|
| **Order_id** | **Order_id** | **Order_id** | **Product_id** | **Aisle_id** | **Department_id** |
| **Product_id** | **Product_id** | Buyer | Product_name | aisle_name | Department_name |
| Add_to_cart_order | Add_to_cart_order | The day of week | **Aisle_id** | | |
| Reordered | Reordered | Days since prior order, etc. | **department_id** | | |

Table 1. Dataset Descriptions

# Dataset

- The anonymized dataset includes over 3 million grocery orders samples from more than 200,000 users. For each user, between 4 and 100 of their orders are given, with the sequence of products purchased in each order.

- Prior dataset gives the past behaviors of a user, while the train and test dataset give the future behaviors that we would like to predict.

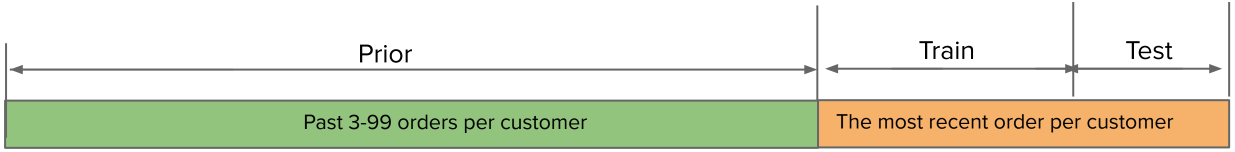


Figure 1. Train / Test Split

# Methods - Feature Engineering

## User Features

1. # User orders in the prior dataset
2. Sum of days since prior order
3. Avg. days since prior order
4. # Reorders / # purchases after the first order
5. # Items that the user has purchased
6. # Distinct products that the user has purchased
7. Avg. number of items per order

## Item Features

1. #Tmes the user purchased the item
2. #Times the item has been purchased
3. #Times the item has been reordered
4. # Unique users have purchased this item
5. Proportion of users who have purchased  this item have reordered this product
6. Percentage of all sold items that are reordered
7. Avg. number of orders of users who have purchased this product

## User x Item Features

1. # Times users purchased the item
2. The order number  at which the user first purchased the item
3. The order number at which the user last purchased the item
4. Avg.  position in the cart
5. # Purchases of this product / Total orders
6. User total order - user last order number
7. # Times the product was purchased / # orders from the first purchase to the last purchase of the product

# Method - Modeling

1. **Baseline: Popularity-based bias model** [4]

Basically we model each interaction R[u,i] as a combination of global, item, and user bias.

- Global bias

$$\mu = \frac{\sum_{u,i} R[u,\ i]}{|R| + \beta_g}$$

where |*R*| is the number of all purchasing counts records and *β* is the damping value

- Item bias

$$b[i] = \frac{\sum_{u} R[u,\ i] - \mu}{|R[:,\ i]| - \beta_i}$$

where |*R[:, i]*| is the number of purchasing counts records associated with item *i*

- User bias

$$b[u] = \frac{\sum_{i} R[u,\ i] - \mu - b[i]}{|R[u,\ :]| - \beta_u}$$

where |*R[u, :]*| is the number of purchasing counts records associated with user *u*

- R[u,i] is the probability that the (user_id, product_id) pair is reordered.
- F1-score: 0.1736

instacart

# **Method** - Modeling

## 2. XGBoost

- XGBoost is an optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**.

- Training:
    - Prepare a train data frame that merge the future orders (train & test) with prior orders (prior).
    - Generate a label “Reordered” for each (user_id, product_id) pairs to indicate whether the pair was reordered or not (1/0).
    - Train xgboost for binary classification.

- Best hyperparameter combination:
    - {‘*eta*’: 0.1, ''*colsample_bytree*': 1, '*gamma*': 1, '*max_depth*': 15, ‘*min_child_weight*’: 10, '*subsample*': 0.76, ‘*alpha*’:2e-05, ‘*scale_pos_weight*’: 10, ‘*lambda*’: 10}
    - F1 score: 0.3778

instacart

# Method - Modeling

## 3. LightGBM

- LightGBM is a gradient boosting framework that uses tree based learning algorithms.
  - Faster training speed and higher efficiency.
  - Lower memory usage.
  - Better accuracy.
  - Support of parallel, distributed, and GPU learning.
  - Capable of handling large-scale data.

- The training procedure is the same as XGBoost.

- Best hyperparameter combination:
  - {'*colsample_bytree*': 1, '*learning_rate*': 0.1, '*max_depth*': -1, '*n_estimators*': 100, '*num_leaves*': 100, '*subsample*': 0.8}
  - F1 score: 0.2976

instacart

# Method - Evaluation

- In this project, we adopted F1 Score (1), the harmonic mean of precision and recall, as our evaluation metric.

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1}$$

$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \text{ or } \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

(2)

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \text{ or } \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

- To calculate F1 Score, we converted the predicted probabilities into binary labels using a threshold value.

- We used the grid search method to find the optimal threshold.

instacart

# Runtime Before Optimization

### Feature Engineering

- Data loading: 7.613 secs
- Product features engineering: 60.344 secs
- User features engineering: 37.403 secs
- User - Product features engineering: 43.979 secs

### Modeling - XGBoost

- Training: 26 secs
- Hyper param tuning: 10829.683 secs (~ 3 hrs)

### Modeling - Baseline

1865 secs (~ 30 min)

### Modeling - LightGBM

- Training: 5.19 secs
- Hyper param tuning: 4433.241 secs (~ 1.2 hrs)

# Method - Optimization

In this project, we explored a range of performance optimization techniques covered in class.

## Feature Engineering Optimization

- Multiprocessing Pool
- Reduce dataframe memory usage
- Vectorization
- Indexing before merging or joining dataframes

## XGboost Optimization

- Reduce dataframe memory usage

```
Memory usage of properties dataframe is : 1010.7310791015625  MB

___MEMORY USAGE AFTER COMPLETION:___
Memory usage is:  315.8535461425781  MB
This is  31.25000830323135 % of the initial size
```

- *n_jobs*, *nthread* parameter
- Halving grid search
- MPI for hyperparameter tuning

instacart

# Feature Engineering Optimization

- Multiprocessing Pool

```
cpu_cnt = os.cpu_count()
with Pool(cpu_cnt) as p:
    pos = p.map(merge_mul_partial, np.array_split(df_right, cpu_cnt))

prior_orders = pd.concat(list(pos))
```

- Vectorization

```
prior_orders[po_col] = prior_orders.groupby(['user_id', 'product_id']).cumcount() + 1
```

- Reduce dataframe memory usage:

  Converting columns to data types with smaller memory usage

```
prod[['product_id', col_2nd]] = prod[['product_id', col_2nd]].apply(np.uint16)
prod[[col_tot, col_1st]] = prod[[col_tot, col_1st]].apply(np.uint32)
prod[prod.select_dtypes(np.float64).columns] = prod.select_dtypes(np.float64).astype(np.float16)
```

```
Original memory usage of newly generated product features is 3.032 MBs.
Optimized memory usage of newly generated product features is 0.948 MBs.
```

- Indexing before merging or joining dataframes

```
d, p = data_opt.set_index('product_id'), prod_opt.set_index('product_id')
u = users_opt.set_index('user_id')
data_opt = d.join(p, how = 'inner').reset_index().set_index('user_id').join(u, how = 'inner').reset_index()
```

# Hyperparameter Tuning Optimization - (XGBoost)

## Halving GridSearch

- sklearn.model_selection.HalvingGridSearchCV
- 1039.029 seconds
- ~ 17.31 minutes
- ~ 10x faster than GridSearchCV

```
Best parameters found:  {'colsample_bytree': 1, 'gamma': 1, 'max_depth': 15, 'subsample': 0.76, 'n_estimators': 9}
Best f1:  0.291358862843743 74
Hyperparamter tuning spent 1039.02920794487 seconds
```

## MPI

- Itertools.product
- 1593.375 secs
- ~ 26 min
- ~ 7x faster than GridSearchCV
- CONS: require large RAM

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
N = comm.Get_size()

# Hyperparamter sets
colsample_bytree = [0.95, 1]
subsample = [0.76, 0.8]
max_depth = [10, 15]
gamma = [0.1, 1]

params = [colsample_bytree, subsample, max_depth, gamma]
params_grids = list(itertools.product(*params))
n_params = len(params_grids)
n_param_rank = n_params // N
```

# Optimization Results

## Before optimization ~ 4.8 hrs

- **Feature engineering**: 149.339 secs
  - Data loading: 7.613 secs, 1214.783 MBs
  - Product feature engineering: 60.344 secs, 3.032 MBs
  - User feature engineering: 37.403 secs, 18.879 MBs
  - User - Product feature engineering: 43.979 secs, 1307.22 MBs
- **Modeling:** 17,158.431 secs
  - Baseline: 1865 secs (~ 30 mins)
  - XGBoost:
    - Training: 26 secs
    - Hyperparameter tuning: 10829.683 secs (~ 3 hrs)
  - LightGBM:
    - Training: 5.19 secs
    - Hyperparameter tuning: 4433.241 secs (~ 1.2 hrs)

## After Optimization ~ 1 hr

- **Feature engineering**: 108.77 secs
  - Data loading: 6.45 secs, 303.696 MBs
  - Product feature engineering: 46.905 secs, 0.948 MBs
  - User feature engineering: 27.361 secs, 5.113 MBs
  - User - Product feature engineering: 27.361 secs, 5.113 MBs
- **Modeling:** 3,426.818 secs
  - Baseline: 1865 secs
  - XGBoost:
    - Training: 53 secs
    - Hyperparameter tuning: 1039.02 secs (~ 17 mins)
  - LightGBM:
    - Training: 5.248 secs
    - Hyperparameter tuning: 464.55 secs (~ 7 mins)

# Conclusion and Future Work

- Both XGBoost and LightGBM achieved significantly better result than the popularity baseline.
- XGBoost is the best model based on F1 score while LightGBM requires less training time at the cost of accuracy.
- Using optimization techniques, the total runtime has decreased by 3.8 hrs.
- The runtime could be further improved by having more computing resources

# Q & A

# References


[1] https://www.kaggle.com/c/instacart-market-basket-analysis/overview

[2] https://towardsdatascience.com/10-tips-tricks-for-worKing-with-large-datasets-in-machine-learning-7065f1d6a802

[3] https://tech.instacart.com/3-million-instacart-orders-open-sourced-d40d29ead6f2

[4] https://newclasses.nyu.edu/access/content/group/8bce5f3c-c1f9-48a9-9f2a-7abb59e6b9a5/Slides/Week%2010.1%20Recommender%20systems.pdf

[5] https://arxiv.org/pdf/1301.3781.pdf

[6] https://link.springer.com/article/10.1057/dbm.2012.17

[7] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.HalvingGridSearchCV.html


instacart