Deep Learning Based Product Reordering Prediction

Technical Documentation

Table of Contents

Feature Generation

1. Data Integration Process

The feature engineering process begins with merging multiple datasets to create a comprehensive user-product interaction view:

```python
# Product enrichment with aisle and department information
products = pd.merge(products, aisles, on='aisle_id', how='left')
products = pd.merge(products, departments, on='department_id', how='left')

# Order-product integration with user information
orders_prior = pd.merge(order_products_prior, orders, on='order_id', how='left')
orders_full = pd.merge(orders_prior, products, on='product_id', how='left')
```

2. Feature Categories

2.1 User-Level Features

Generated by aggregating user behavior across all orders:

- `user_total_orders`: Maximum order number per user (indicates user engagement level)
- `user_avg_days_between_orders`: Average days between consecutive orders (ordering frequency pattern)
- `user_days_since_last_order`: Days since the user's most recent order (recency indicator)

python

```python
user_features = df.groupby('user_id').agg(
    user_total_orders=('order_number', 'max'),
    user_avg_days_between_orders=('days_since_prior_order', 'mean'),
    user_days_since_last_order=('days_since_prior_order', 'last')
).reset_index()
```

2.2 Product-Level Features

Aggregated product popularity metrics across all users:

- `prod_total_orders`: Total times the product was ordered (product popularity)
- `prod_reorders`: Total reorder instances for the product (reorder appeal)
- `prod_reorder_ratio`: Ratio of reorders to total orders (product loyalty metric)

python

```python
product_features = df.groupby('product_id').agg(
    prod_total_orders=('reordered', 'count'),
    prod_reorders=('reordered', 'sum'),
    prod_reorder_ratio=('reordered', 'mean')
).reset_index()
```

2.3 User-Product Interaction Features

Captures specific user-product relationship dynamics:

- `up_orders`: Number of times specific user ordered this product
- `up_first_order`: Order sequence when user first tried this product
- `up_last_order`: Most recent order sequence for this product
- `up_reorders`: Number of reorder instances for this user-product pair

python

```python
user_product_features = df.groupby(['user_id', 'product_id']).agg(
    up_orders=('order_number', 'count'),
    up_first_order=('order_number', 'min'),
    up_last_order=('order_number', 'max'),
    up_reorders=('reordered', 'sum')
).reset_index()
```

2.4 Derived Features

Computed ratios providing normalized interaction metrics:

- `up_order_rate`: User-product order frequency relative to user's total activity
  - Formula: `up_orders / user_total_orders`
  - Indicates product importance in user's shopping basket

3. Data Quality and Distribution Analysis

3.1 Missing Value Treatment

- **Primary Gap**: `days_since_prior_order` contains nulls for first-time orders
- **Solution**: Fill missing values with 0, representing new customer baseline

3.2 Skewness Correction

Statistical analysis revealed high skewness in numerical features:

Original Distribution Issues:

- Most features showed significant positive skewness (>2.0)
- High kurtosis indicating extreme outliers
- Non-normal distributions affecting model performance

Log Transformation Applied:

```python
log_transform_cols = [
    'up_orders', 'up_first_order', 'up_last_order',
    'up_reorders', 'up_order_rate', 'user_total_orders',
    'prod_total_orders', 'prod_reorders'
]

for col in log_transform_cols:
    final_features_new[col] = np.log1p(final_features_new[col])
```

Post-Transformation Results:

- Reduced skewness to acceptable ranges (typically <2.0)
- Improved kurtosis values
- Better feature distribution for neural network training

3.3 Categorical Encoding

- **Method**: Label Encoding for categorical variables
- **Variables Encoded**: product_name, aisle, department

- **Rationale**: Maintains ordinal relationships while enabling neural network processing

4. Class Imbalance Handling

Original Distribution:

- Class 0 (No Reorder): ~89%
- Class 1 (Reorder): ~11%

Undersampling Strategy:

- **Method**: RandomUnderSampler
- **Sampling Ratio**: 0.666 (approximately 2:3 ratio)
- **Rationale**: Balance classes while preserving sufficient training data

---

Model Architecture

1. Neural Network Design

1.1 Architecture Overview

Deep Feedforward Neural Network with regularization:

```
Input Layer (11 features)
    ↓
Dense Layer 1: 256 neurons + ReLU + BatchNorm + Dropout(0.3)
    ↓
Dense Layer 2: 128 neurons + ReLU + BatchNorm + Dropout(0.25)
    ↓
Dense Layer 3: 64 neurons + ReLU + BatchNorm + Dropout(0.2)
    ↓
Dense Layer 4: 32 neurons + ReLU + BatchNorm + Dropout(0.15)
    ↓
Output Layer: 1 neuron + Sigmoid
```

1.2 Layer-by-Layer Specification

Hidden Layer 1:

- **Neurons**: 256
- **Activation**: ReLU (for non-linearity)
- **BatchNormalization**: Stabilizes training and accelerates convergence
- **Dropout**: 30% (prevents overfitting on complex patterns)

Hidden Layer 2:

- **Neurons**: 128 (progressive reduction)
- **Dropout**: 25% (slightly reduced as network narrows)

Hidden Layer 3:

- **Neurons**: 64
- **Dropout**: 20%

Hidden Layer 4:

- **Neurons**: 32
- **Dropout**: 15% (minimal dropout near output)

Output Layer:

- **Neurons**: 1
- **Activation**: Sigmoid (binary classification probability)

1.3 Training Configuration

Optimizer:

- **Type**: Adam optimizer
- **Learning Rate**: 0.001
- **Rationale**: Adaptive learning rate with momentum for stable convergence

Loss Function:

- **Type**: Binary Cross-Entropy
- **Justification**: Optimal for binary classification problems

Metrics Monitored:

- Accuracy: Overall classification performance
- Precision: True positive rate among predictions
- Recall: True positive rate among actual positives
- AUC: Area under ROC curve for threshold-independent evaluation

Regularization Techniques:

1. **Dropout**: Prevents overfitting by randomly deactivating neurons
2. **Batch Normalization**: Reduces internal covariate shift
3. **Early Stopping**: Prevents overfitting by monitoring validation loss

Training Parameters:

- **Batch Size**: 512 (balance between memory efficiency and gradient stability)
- **Maximum Epochs**: 50
- **Early Stopping Patience**: 10 epochs
- **Validation Split**: Separate validation set for unbiased evaluation

2. Data Preprocessing for Model Input

2.1 Feature Scaling

```python
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train_numeric)
x_val_scaled = scaler.transform(x_val_numeric)
x_test_scaled = scaler.transform(x_test_numeric)
```

Rationale: Neural networks require normalized inputs for optimal performance

2.2 Train-Validation-Test Split

- **Training**: 70% of undersampled data
- **Validation**: 10% (for model selection and hyperparameter tuning)
- **Test**: 20% (for final unbiased evaluation)
- **Stratification**: Maintains class distribution across splits

---

Evaluation Results

1. Model Performance Metrics

1.1 Classification Performance

Target Achievement:

- **F1-Score Requirement**: >0.65 ✓
- **Actual Performance**: Achieved target on validation data

Key Metrics:

- **Accuracy**: Overall prediction correctness
- **Precision**: Minimizes false positive recommendations
- **Recall**: Captures actual reorder instances
- **F1-Score**: Balanced precision-recall performance
- **ROC-AUC**: Discrimination ability across all thresholds

1.2 Confusion Matrix Analysis

The confusion matrix provides detailed breakdown of:

- **True Positives**: Correctly predicted reorders
- **True Negatives**: Correctly predicted non-reorders
- **False Positives**: Incorrectly predicted reorders (marketing cost)
- **False Negatives**: Missed reorder opportunities (lost revenue)

2. Business Validation Results

2.1 Personalized Product Recommendation

High-Confidence Predictions (Probability > 0.7):

- Identified top products with highest reorder likelihood
- Recommendation accuracy measured for high-probability predictions
- Focus on products with sufficient frequency (>5 occurrences)

Key Insights:

- Products with consistently high reorder probabilities across users
- Correlation between predicted and actual reorder rates
- Recommendation system effectiveness validation

2.2 Inventory Management

Demand Forecasting by Department/Aisle:

- Aggregated reorder predictions for inventory planning
- Comparison of predicted vs. actual demand
- Demand accuracy calculation: `1 - |predicted - actual| / actual`

Performance Metrics:

- Overall demand forecasting accuracy across categories
- Identification of high-demand departments/aisles
- Inventory optimization insights

2.3 Customer Retention Analysis

Churn Risk Assessment:

- Low reorder probability (<0.3) indicates potential churn
- Customer-level aggregation of reorder probabilities

- Churn prediction accuracy validation

Results:

- Percentage of customers identified as high churn risk
- Accuracy of churn risk predictions
- Customer retention strategy insights

2.4 Marketing Optimization

Customer Segmentation by Reorder Intent:

- **High Intent (≥0.8)**: Upsell opportunities
- **Medium Intent (0.5-0.8)**: Nurturing campaigns
- **Low Intent (0.2-0.5)**: Re-engagement strategies
- **Very Low Intent (<0.2)**: Win-back campaigns

Segment Performance:

- Actual conversion rates by predicted segment
- Marketing campaign targeting effectiveness
- ROI optimization through segment-specific strategies

3. Model Validation and Robustness

3.1 Generalization Performance

- Consistent performance across training, validation, and test sets
- No significant overfitting observed with regularization techniques
- Stable predictions across different user and product segments

3.2 Feature Importance and Interpretability

- User-product interaction features showed highest predictive power
- Product-level features contributed to general reorder patterns
- User-level features provided personalization context

3.3 Deployment Readiness

Model Artifacts Generated:

- `instacart_model.keras`: Trained neural network model
- `scaler.pkl`: Feature scaling parameters
- Encoder mappings for categorical variables

Production Considerations:

- Consistent preprocessing pipeline
- Scalable prediction infrastructure via Streamlit interface
- Real-time prediction capability for individual user-product pairs

---

Business Impact Analysis

1. Revenue Optimization

- **Precision-focused**: Minimize wasted marketing spend on unlikely reorders
- **Recall-balanced**: Capture sufficient reorder opportunities
- **Probability-based**: Enable threshold tuning for different business objectives

2. Operational Efficiency

- **Inventory Planning**: Data-driven demand forecasting
- **Customer Targeting**: Segment-specific marketing strategies
- **Resource Allocation**: Focus on high-value customer-product combinations

3. Strategic Insights

- **Product Portfolio**: Identify high-loyalty vs. trial products
- **Customer Behavior**: Understand reorder patterns and preferences
- **Market Dynamics**: Predict category-level demand trends

This documentation provides comprehensive coverage of the technical implementation, validation results, and business applications of the deep learning-based product reordering prediction system.