Model Training Pipeline Documentation

Overview

This document provides a detailed description of the complete training pipeline for the Smart List Project - Product Recommendation Model. The model predicts the next product category a client is likely to purchase based on their historical purchase patterns and demographic information.

Table of Contents

---

1. Data Source and Identification

1.1 Source Table

- **Table Name**: `dl_tenants_daas.us_wealth_management.wealth_management_client_metrics`
- **Platform**: Databricks Spark SQL
- **Data Type**: Client demographic data

1.2 Record Identification

The pipeline loads all records from the source table without initial filtering:

```
df_raw =
spark.table("dl_tenants_daas.us_wealth_management.wealth_management_client_
metrics")
```

---

2. Data Preprocessing

2.1 Product Category Transformation

The raw data contains multiple product classification fields:

- `prod_lob` (Product Line of Business)

- `sub_product_level_1`
- `sub_product_level_2`

These are consolidated into a single `product_category` field with the following categories:

- **LIFE_INSURANCE**: Life insurance products (VLI, WL, UL/IUL, TERM, etc.)
- **RETIREMENT**: Retirement products (401K, IRA, 403B, SEP, etc.)
- **INVESTMENT**: Investment products (Brokerage, Advisory, Direct)
- **NETWORK_PRODUCTS**: Network-related products
- **DISABILITY**: Disability insurance products
- **HEALTH**: Health insurance products
- **OTHER**: All other products

Transformation Logic: Hierarchical rules applied in order:

1. Check `prod_lob` first
2. Then check `sub_product_level_1`
3. Finally check `sub_product_level_2` with pattern matching

2.2 Data Filtering

2.2.1 Required Fields Filter

Keep only records with:

- Non-null `cont_id` (Client ID)
- Non-null `register_date` (Event date)
- Non-null `product_category` (After transformation)

Selected Columns:

- Client identifiers: `cont_id`
- Product information: `product_category`, `register_date`
- Financial metrics: `acct_val_amt`, `face_amt`, `cash_val_amt`, `wc_total_assets`
- Asset mix: `wc_assetmix_stocks`, `wc_assetmix_bonds`, `wc_assetmix_mutual_funds`, `wc_assetmix_annuity`, `wc_assetmix_deposits`, `wc_assetmix_other_assets`
- Demographics: `psn_age`, `client_seg`, `client_seg_1`, `aum_band`
- Channel information: `channel`, `agent_segment`, `branchoffice_code`
- Policy status: `policy_status`

2.2.2 Policy Status Filter

Filter to keep only **Active** policies:

```
df_events = df_events.filter(F.col("policy_status") == "Active")
```

2.2.3 Optional Sampling

For computational efficiency, apply random sampling:

- **Sample Fraction**: 0.2 (20% of data)
- **Seed**: 42 (for reproducibility)
- **Sampling Method**: Without replacement

Result: ~48,978,063 event rows after sampling

2.3 Event Ordering

Events are ordered chronologically per client:

1. Convert `register_date` to timestamp: `register_ts`
2. Create window function partitioned by `cont_id`, ordered by `register_ts`
3. Assign `event_idx` (row number) to each event per client

Purpose: Establish temporal sequence for each client's purchase history.

---

3. Feature Engineering

3.1 Product Vocabulary Building

Build a mapping between product categories and numeric IDs:

1. Extract all unique `product_category` values from the entire dataset
2. Sort alphabetically
3. Create mappings:
    - `prod2id`: Product name → ID (starting from 1, 0 reserved for padding)
    - `id2prod`: ID → Product name
4. **Vocabulary Size**: 7 product categories

Example:

```
prod2id = {
    'DISABILITY': 1,
    'HEALTH': 2,
    'INVESTMENT': 3,
    'LIFE_INSURANCE': 4,
    'NETWORK_PRODUCTS': 5,
    'OTHER': 6,
    'RETIREMENT': 7
}
```

3.2 Sequence Building and Deduplication

3.2.1 Group Events by Client

- Group all events by `cont_id`
- Sort events by `event_idx` within each client

- Extract product category sequence per client

3.2.2 Minimum Events Filter

Keep only clients with at least `MIN_EVENTS` (default: 2) products:

- Ensures at least 1 history item + 1 label per training example
- **Result**: ~388,326 users with sufficient history

3.3 Sliding Window Example Generation

For each client sequence, create multiple training examples using a sliding window approach:

Process:

1. For each position `i` in sequence (starting from index 1):
   - **History**: Last `MAX_SEQ_LEN` products before position `i`
   - **Label**: Product at position `i`
2. Convert products to numeric IDs using `prod2id` mapping
3. Pad history to `MAX_SEQ_LEN` if shorter (left-pad with 0)

Example:

- Sequence: `[RETIREMENT, LIFE_INSURANCE, INVESTMENT]` (IDs: [7, 4, 3])
- Examples generated:
  - History: `[7]`, Label: `4` (LIFE_INSURANCE)
  - History: `[7, 4]`, Label: `3` (INVESTMENT)

3.4 History-Derived Features

For each training example, compute features from the history sequence:

3.4.1 Sequence Statistics

- **seq_len**: Length of history sequence (before padding)
- **last_1**: Product ID of most recent product in history
- **last_2**: Product ID of second most recent product (0 if sequence length < 2)
- **unique_prior**: Number of unique products in history
- **num_switches**: Number of product transitions in history

3.4.2 Frequency Features

For each product category (1 to NUM_CLASSES), compute:

- **freq_{i}**: Count of product ID `i` in history sequence

Example: If history contains `[RETIREMENT, RETIREMENT, LIFE_INSURANCE]`:

- `freq_7` = 2 (RETIREMENT appears twice)
- `freq_4` = 1 (LIFE_INSURANCE appears once)
- `freq_1` through `freq_6` = 0

Total Frequency Features: 7 (one per product category)

3.5 Padded History Sequence Features

Convert variable-length history sequences to fixed-length features:

- **MAX_SEQ_LEN**: 10 (maximum history length)
- **Padding**: Left-pad with 0s if sequence shorter than MAX_SEQ_LEN
- **Truncation**: Keep only last MAX_SEQ_LEN products if sequence longer

Features Created: `hist_0`, `hist_1`, ..., `hist_9`

- Each represents a position in the padded history sequence
- Value is product ID (0 for padding)

Example:

- History: `[7, 4, 3]` (length 3)
- Padded: `[0, 0, 0, 0, 0, 0, 0, 7, 4, 3]`
- Features: `hist_0=0, hist_1=0, ..., hist_7=7, hist_8=4, hist_9=3`

3.6 Static Client Features

Join with the most recent client snapshot (latest `register_ts` per client):

3.6.1 Financial Features

- `acct_val_amt`: Account value amount
- `face_amt`: Face amount
- `cash_val_amt`: Cash value amount
- `wc_total_assets`: Total assets
- `wc_assetmix_stocks`: Stock allocation
- `wc_assetmix_bonds`: Bond allocation
- `wc_assetmix_mutual_funds`: Mutual fund allocation
- `wc_assetmix_annuity`: Annuity allocation
- `wc_assetmix_deposits`: Deposit allocation
- `wc_assetmix_other_assets`: Other assets

3.6.2 Demographic Features

- `psn_age`: Person age

3.6.3 Categorical Features

- `client_seg`: Client segment
- `client_seg_1`: Client segment level 1
- `aum_band`: Assets under management band
- `channel`: Sales channel
- `agent_segment`: Agent segment
- `branchoffice_code`: Branch office code

Join Method: Left join to preserve all training examples

3.7 Missing Value Handling

3.7.1 Numeric Features

- Fill missing values with **0**
- Applied to all numeric columns except: `label`, `seq_len`, `last_1`, `last_2`, `unique_prior`, `num_switches`

3.7.2 Categorical Features

- Compute mode (most frequent value) for each categorical column
- Fill missing values with mode
- If mode is null, use "UNKNOWN"

Categorical Columns: `client_seg`, `client_seg_1`, `aum_band`, `channel`, `agent_segment`, `branchoffice_code`

3.8 Categorical Encoding

Convert categorical string values to integer indices:

1. For each categorical column:
    - Extract all unique values
    - Sort alphabetically
    - Create mapping: value → index (0-based)
2. Create new column: `{categorical_col}_idx`
3. Map values using UDF (User Defined Function)

Example:

- `channel` values: `["Direct", "Advisor", "Online"]`
- Mapping: `{"Direct": 0, "Advisor": 1, "Online": 2}`
- New column: `channel_idx` with integer values

Categorical Columns Encoded: 6 (each gets an `_idx` column)

3.9 Final Feature Set

Total Features: 39

Feature Categories:

1. **History Sequence Features** (10): `hist_0` through `hist_9`
2. **Sequence Statistics** (5): `seq_len`, `last_1`, `last_2`, `unique_prior`, `num_switches`
3. **Frequency Features** (7): `freq_1` through `freq_7`
4. **Financial Features** (11): Account values, face amount, cash value, total assets, asset mix (6 categories)
5. **Demographic Features** (1): `psn_age`
6. **Categorical Index Features** (6): `client_seg_idx`, `client_seg_1_idx`, `aum_band_idx`, `channel_idx`, `agent_segment_idx`, `branchoffice_code_idx`

---

4. Sequence Building

4.1 Training Example Structure

Each training example represents a prediction task:

- **Input**: Client's purchase history up to a point in time
- **Output**: Next product category purchased

4.2 Example Generation Process

1. **Client Sequences**: Each client has a chronological sequence of product purchases
2. **Sliding Window**: For each position in the sequence (except the first):
   - Use all previous products as history
   - Use current product as label
3. **Multiple Examples per Client**: Clients with longer sequences generate more examples

Example Client Journey:

- Client purchases: `[RETIREMENT, RETIREMENT, LIFE_INSURANCE, INVESTMENT]`
- After deduplication: `[RETIREMENT, LIFE_INSURANCE, INVESTMENT]`
- Training examples:
  - Example 1: History=`[RETIREMENT]`, Label=`LIFE_INSURANCE`
  - Example 2: History=`[RETIREMENT, LIFE_INSURANCE]`, Label=`INVESTMENT`

4.3 Data Conversion

Convert from Spark DataFrame to Pandas DataFrame:

- **Reason**: LightGBM requires Pandas/NumPy input
- **Caching**: Cache Spark DataFrames before conversion for performance
- **Columns Selected**: `cont_id`, `label`, and all `model_feature_cols`

---

5. Train/Validation/Test Split

5.1 Split Strategy

- **Method**: Random split (stratified by label not used, but could be added)
- **Proportions**:
  - **Training**: 80% (TRAIN_FRAC = 0.8)
  - **Validation**: 10% (VAL_FRAC = 0.1)
  - **Test**: 10% (TEST_FRAC = 0.1)
- **Seed**: 42 (for reproducibility)

5.2 Split Implementation

```
train_spark, val_spark, test_spark = examples_full.randomSplit(
    [TRAIN_FRAC, VAL_FRAC, TEST_FRAC],
    seed=RANDOM_SEED
)
```

5.3 Data Volumes

After split (approximate):

- **Training**: ~464,400 examples
- **Validation**: ~57,875 examples
- **Test**: ~58,345 examples

5.4 Label Remapping

Convert labels from product IDs (1-7) to zero-based indices (0-6) for LightGBM:

- Create `label_map`: Original product ID → Zero-based index
- Create new column `label0` in all splits
- Update `num_class` parameter to match actual number of classes

Example:

- Original labels: `[1, 2, 3, 4, 5, 6, 7]`
- Mapped labels: `[0, 1, 2, 3, 4, 5, 6]`
- `num_classes`: 7

5.5 Final Data Preparation

1. **Fill Missing Values**: Fill any remaining NaN values with 0
2. **Data Types**: Ensure all features are numeric (integers or floats)
3. **Feature Ordering**: Maintain consistent feature column order across all splits

---

6. Model Training

6.1 Model Algorithm

- **Algorithm**: LightGBM (Gradient Boosting Decision Trees)
- **Task**: Multiclass Classification
- **Number of Classes**: 7 product categories

6.2 Training Parameters

```
LGB_PARAMS = {
    "objective": "multiclass",
    "num_class": 7,
    "metric": "multi_logloss",
    "boosting_type": "gbdt",
    "learning_rate": 0.05,
    "num_leaves": 64,
    "min_data_in_leaf": 50,
    "feature_fraction": 0.8,
    "subsample": 0.8,
    "subsample_freq": 1,
    "lambda_l2": 2.0,
    "verbosity": -1
}
```

Parameter Descriptions:

- **objective**: "multiclass" for multi-class classification
- **num_class**: Number of product categories (7)
- **metric**: "multi_logloss" (multi-class log loss)
- **learning_rate**: 0.05 (conservative learning rate)
- **num_leaves**: 64 (tree complexity)
- **min_data_in_leaf**: 50 (minimum samples per leaf)
- **feature_fraction**: 0.8 (use 80% of features per tree)
- **subsample**: 0.8 (use 80% of data per tree)
- **lambda_l2**: 2.0 (L2 regularization)

6.3 Training Configuration

- **Number of Boost Rounds**: 2000 (maximum iterations)
- **Early Stopping**: 50 rounds (stop if validation loss doesn't improve)
- **Validation Set**: Used for early stopping and monitoring

6.4 Training Process

1. **Create LightGBM Datasets**:
   - Training dataset: `train_pd[feature_cols_final]` with labels `train_pd["label0"]`
   - Validation dataset: `val_pd[feature_cols_final]` with labels `val_pd["label0"]`
2. **Train Model**:
   - Monitor both training and validation loss
   - Stop early if validation loss doesn't improve for 50 rounds
   - Save best model based on validation performance

---

8. Model Registration

8.1 MLflow Integration

The trained model is registered using MLflow for model versioning and deployment:

```
mlflow.lightgbm.log_model(
    model,
    artifact_path="final_lgbm_multiclass_model",

registered_model_name="eda_smartlist.models.final_lgbm_multiclass_model",
    signature=signature,
    input_example=input_example
)
```

8.2 Model Artifacts

Registered Components:

1. **Model File**: LightGBM model binary
2. **Model Signature**: Input/output schema
3. **Input Example**: Sample input data for reference
4. **Metadata**: Training parameters, metrics, etc.

8.3 Model Version

Create an alias for easy model access:

- **Model Name**: `eda_smartlist.models.final_lgbm_multiclass_model`
- **Version**: 1

8.4 Artifact Storage

Additional artifacts saved separately:

- **artifacts.pkl**: Contains:
  - `prod2id`: Product to ID mapping
  - `id2prod`: ID to product mapping
  - `label_map`: Label remapping dictionary
  - `num_classes`: Number of classes
  - `feature_cols`: List of feature column names
  - `max_seq_len`: Maximum sequence length (10)
  - `categorical_mappings`: Categorical encoding mappings

Storage Location: `/Workspace/Users/rajesh.patil@equitable.com/Final_model_files/`

---

9. Key Parameters

9.1 Data Processing Parameters

Parameter	Value	Description
`SAMPLE_FRACTION`	0.2	Fraction of data to sample (None for full dataset)
`MIN_EVENTS`	2	Minimum events per client to create examples
`MAX_SEQ_LEN`	10	Maximum history length for features

9.2 Train/Test Split Parameters

Parameter	Value	Description

Parameter	Value	Description
`TRAIN_FRAC`	0.8	Training set proportion (80%)
`VAL_FRAC`	0.1	Validation set proportion (10%)
`TEST_FRAC`	0.1	Test set proportion (10%)
`RANDOM_SEED`	42	Random seed for reproducibility

9.3 Model Parameters

Parameter	Value	Description
`NUM_BOOST_ROUND`	2000	Maximum number of boosting rounds
`EARLY_STOP`	50	Early stopping rounds
`learning_rate`	0.05	Learning rate for gradient boosting
`num_leaves`	64	Maximum tree leaves
`min_data_in_leaf`	50	Minimum samples per leaf
`feature_fraction`	0.8	Feature sampling ratio
`subsample`	0.8	Data sampling ratio
`lambda_l2`	2.0	L2 regularization

---

10. Data Flow Summary

10.1 Source to Training Examples

```
Source Table (48M+ rows)
    ↓
Product Category Transformation
    ↓
Filter: Active policies, non-null fields
    ↓
Optional Sampling (20%)
    ↓
Event Ordering (by register_date per client)
    ↓
Sequence Building (group by client)
    ↓
Deduplicate Consecutive Products
    ↓
Filter: Clients with ≥2 events
    ↓
Sliding Window Example Generation
    ↓
Feature Engineering
    ↓
```

```
Train/Val/Test Split (80/10/10)
    ↓
Training Examples (464K+ examples)
```

10.2 Feature Categories

1. **Temporal Features**: History sequence (hist_0 to hist_9)
2. **Sequence Statistics**: Length, last products, transitions
3. **Frequency Features**: Product purchase frequencies
4. **Financial Features**: Account values, asset allocations
5. **Demographic Features**: Age, client segments
6. **Categorical Features**: Channel, agent segment, branch office

---

11. Key Design Decisions

11.1 Why Sliding Window?

- **Captures Evolution**: Each example represents a point in the client's journey
- **Maximizes Data**: One client with N products generates N-1 examples
- **Temporal Context**: Model learns from sequences of different lengths

11.2 Why LightGBM?

- **Handles Mixed Features**: Works well with numeric and categorical features
- **Efficiency**: Fast training on large datasets
- **Performance**: Strong results on tabular data
- **Interpretability**: Feature importance and SHAP values available

---

Appendix A: Column Reference

A.1 Source Table Columns Used

Identifiers:

- `cont_id`: Client/Contract ID
- `axa_party_id`: AXA Party ID
- `branchoffice_code`: Branch office code

Product Information:

- `prod_lob`: Product line of business
- `sub_product_level_1`: Sub-product level 1
- `sub_product_level_2`: Sub-product level 2
- `register_date`: Product registration date
- `policy_status`: Policy status (Active/Inactive)

Financial Metrics:

- `acct_val_amt`: Account value amount
- `face_amt`: Face amount
- `cash_val_amt`: Cash value amount
- `wc_total_assets`: Total assets
- `wc_assetmix_stocks`: Stock allocation
- `wc_assetmix_bonds`: Bond allocation
- `wc_assetmix_mutual_funds`: Mutual fund allocation
- `wc_assetmix_annuity`: Annuity allocation
- `wc_assetmix_deposits`: Deposit allocation
- `wc_assetmix_other_assets`: Other assets

Demographics:

- `psn_age`: Person age
- `client_seg`: Client segment
- `client_seg_1`: Client segment level 1
- `aum_band`: Assets under management band

Channel Information:

- `channel`: Sales channel
- `agent_segment`: Agent segment
- `division_name`: Division name
- `branch_name`: Branch name
- `business_city`: Business city
- `business_state_cod`: Business state code

Temporal:

- `business_month`: Business month (YYYYMM format)

Appendix B: Product Category Mapping Rules

B.1 LIFE_INSURANCE

- `prod_lob == "LIFE"`
- `sub_product_level_1` in: `["VLI", "WL", "UL/IUL", "TERM", "PROTECTIVE PRODUCT"]`
- `sub_product_level_2` contains "LIFE"
- `sub_product_level_2` in: `["VARIABLE UNIVERSAL LIFE", "WHOLE LIFE", "UNIVERSAL LIFE", "INDEX UNIVERSAL LIFE", "TERM PRODUCT", "VARIABLE LIFE", "SURVIVORSHIP WHOLE LIFE", "MONY PROTECTIVE PRODUCT"]`

B.2 RETIREMENT

- `prod_lob` in: `["GROUP RETIREMENT", "INDIVIDUAL RETIREMENT"]`
- `sub_product_level_1` in: `["EQUIVEST", "RETIREMENT 401K", "ACCUMULATOR", "RETIREMENT CORNERSTONE", "SCS", "INVESTMENT EDGE"]`
- `sub_product_level_2` contains: "403B", "401", "IRA", or "SEP"

B.3 INVESTMENT

- `prod_lob == "BROKER DEALER"`
- `sub_product_level_1` in: `["INVESTMENT PRODUCT - DIRECT", "INVESTMENT PRODUCT - BROKERAGE", "INVESTMENT PRODUCT - ADVISORY", "DIRECT", "BROKERAGE", "ADVISORY", "CASH SOLICITOR"]`
- `sub_product_level_2` contains: "Investment", "Brokerage", or "Advisory"

B.4 NETWORK_PRODUCTS

- `prod_lob == "NETWORK"`
- `sub_product_level_1 == "NETWORK PRODUCTS"`
- `sub_product_level_2 == "NETWORK PRODUCTS"`

B.5 DISABILITY

- `prod_lob == "OTHERS"` AND `sub_product_level_1 == "HAS"`
- `sub_product_level_2 == "HAS - DISABILITY"`

B.6 HEALTH

- `prod_lob == "OTHERS"` (and not DISABILITY)
- `sub_product_level_2 == "GROUP HEALTH PRODUCTS"`

B.7 OTHER

- All products not matching above rules