AI-Powered Customer Purchase Analysis

by Branislav Njemec

Date: February 19, 2025

Contents

1 Introduction

This report describes the design and implementation of an AI-powered solution for analyzing customer purchase data and generating targeted product recommendations. Our objectives:

- Generate a synthetic dataset with realistic purchase records.
- Identify top-selling products and categories.
- Classify customers based on their purchase behavior.
- Deliver a hybrid recommendation system combining Collaborative and Content-Based filtering, weighted by recent purchases (time-weighted).

2 Data Generation

We create a synthetic dataset with:

- 500 customers
- 50 products
- 5,000 purchase records

Each record is composed of `(Customer ID, Product ID, Product Category, Purchase Amount, Purchase Date)`.

Listing 1: Key code snippet for generating synthetic data.

```
import pandas as pd
import random
import uuid
from datetime import datetime, timedelta

def random_date():
    # Generates a random date within 2023
    return datetime(2023, 1, 1) + timedelta(days=random.randint(0, 365))

data = []
for _ in range(num_purchases):
    customer_id = random.choice(customers)
    product_id = random.choice(products)
    category = random.choice(categories)
    purchase_amount = round(random.uniform(5, 500), 2)
    purchase_date = random_date().strftime("%Y-%m-%d")
    data.append([customer_id, product_id, category, purchase_amount,
        purchase_date])
```

The generated records are stored in `synthetic_purchase_data.csv` for further analysis.

3 Data Analysis

We use the following metrics to understand the dataset:

- **Top Categories & Products:** By frequency count, we identify top 5 categories and top 5 product IDs.
- **Spending Statistics:**
  - *mean* (average purchase amount per customer)
  - *sum* (total purchase amount per customer)
  - *median* (middle purchase amount)

Listing 2: Analyzing top categories, products, and spending metrics.

```
top_categories = df["Product Category"].value_counts().head()
top_products   = df["Product ID"].value_counts().head()
avg_spending   = df.groupby("Customer ID")["Purchase Amount"].agg(["mean",
    "sum", "median"])
```

These insights guide us in understanding purchasing trends.

4 Customer Classification

We segment customers using K-Means on three factors:

1. **Frequency** (number of purchases)
2. **Total Spending**
3. **Preferred Category** (used mostly for labeling each cluster)

Listing 3: K-Means clustering for customer classification.

```
customer_stats = df.groupby("Customer ID").agg(
    frequency=("Product ID", "count"),
    total_spending=("Purchase Amount", "sum"),
    preferred_category=("Product Category", lambda x: x.mode()[0])
).reset_index()

# Normalize numeric columns
for col in ["frequency", "total_spending"]:
    if customer_stats[col].std() == 0:
        customer_stats[col] = 0
    else:
        customer_stats[col] = (
            customer_stats[col] - customer_stats[col].mean()
        ) / customer_stats[col].std()

kmeans = KMeans(n_clusters=3, random_state=42)
customer_stats["Cluster"] = kmeans.fit_predict(
    customer_stats[["frequency", "total_spending"]]
)

cluster_labels = {0: "Low Spenders", 1: "Medium Spenders", 2: "High
   Spenders"}
customer_stats["Segment"] = customer_stats["Cluster"].map(cluster_labels)
```

4.1 K-Means Clustering Diagram

Figure 1 shows a placeholder for how a K-Means scatter plot might look. You can replace it with your own diagram if you plot frequency vs. total spending.

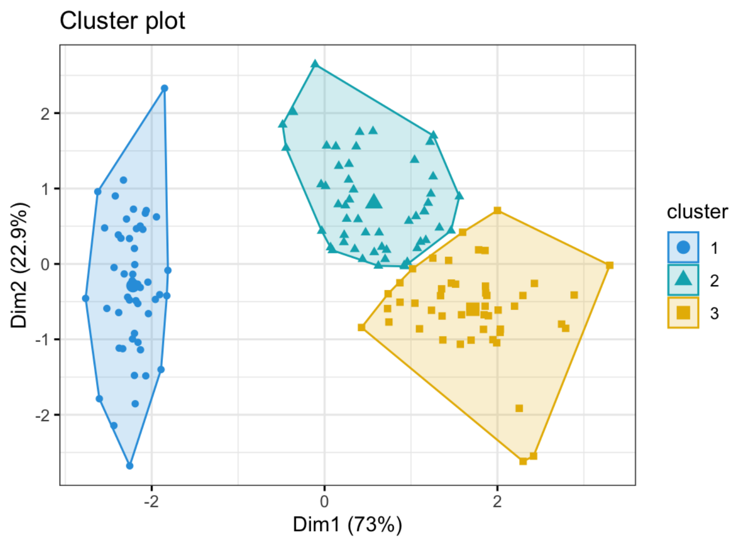

Figure 1: Example K-Means Clustering Visualization (source: Wikipedia Commons)

5 Hybrid Recommendation System

We combine:

1. **Time-weighted Collaborative Filtering**
2. **Content-based Filtering**
3. **Weighted Merge**: 70% Collaborative, 30% Content-based

5.1 Time-weighted Matrix

An exponential decay factor $\exp(-\alpha \times \text{days\_diff})$ is applied, giving more importance to recent purchases.

Listing 4: Building a time-weighted matrix for Collaborative Filtering.

```
def build_time_weighted_matrix(df, alpha=0.01):
    df["Purchase Date"] = pd.to_datetime(df["Purchase Date"])
    latest_date = df["Purchase Date"].max()
    df_weighted = df.copy()
    df_weighted["days_diff"] = (latest_date - df_weighted["Purchase Date"
        ]).dt.days
    df_weighted["weight"] = np.exp(-alpha * df_weighted["days_diff"])
    df_weighted["weighted_amount"] = df_weighted["Purchase Amount"] *
        df_weighted["weight"]

    customer_product_matrix = df_weighted.pivot_table(
        index="Customer ID",
        columns="Product ID",
        values="weighted_amount",
        aggfunc="sum",
```

```
        fill_value=0
    )
    return customer_product_matrix
```

5.2 Merging Collaborative and Content-Based Approaches

We fetch top products from:

- *Collaborative filtering*: Summing up neighbor purchases in the time-weighted matrix.
- *Content-based filtering*: Similar product vectors based on product categories.

We then combine the two lists, remove duplicates, and take 5 final recommendations.

Listing 5: Hybrid recommender with 70-30 split.

```
def recommend_products(df, customer_id, weight_collab=0.7, weight_content
    =0.3, alpha=0.01):
    # 1) Time-weighted matrix
    customer_product_matrix = build_time_weighted_matrix(df, alpha=alpha)

    # 2) Product-Category matrix (standard amounts)
    product_category_matrix = df.pivot_table(
        index="Product ID",
        columns="Product Category",
        values="Purchase Amount",
        fill_value=0
    )

    # If the customer doesn't exist in matrix, return an informational
        message
    if customer_id not in customer_product_matrix.index:
        return ["No recommendation available"]

    # ------------------ Collaborative ------------------
    neigh = NearestNeighbors(metric="cosine", algorithm="brute")
    neigh.fit(customer_product_matrix.to_numpy())

    customer_vector = customer_product_matrix.loc[customer_id].to_numpy().
        reshape(1, -1)
    distances, indices = neigh.kneighbors(customer_vector, n_neighbors=5)
    collab_scores = customer_product_matrix.iloc[indices[0]].sum().
        sort_values(ascending=False)
    collaborative_all = collab_scores.index.tolist()

    # ------------------ Content-Based ------------------
    purchased_products = df[df["Customer ID"] == customer_id]["Product ID"
        ].unique()
    content_based_all = []
    for prod in purchased_products:
        if prod in product_category_matrix.index:
            sim_matrix = cosine_similarity(
                product_category_matrix.loc[prod].values.reshape(1, -1),
                product_category_matrix
            )
            sim_indices = np.argsort(sim_matrix[0])[::-1][1:6]
            content_based_all.extend(product_category_matrix.index[
                sim_indices].tolist())
```

```
    # ------------------ Weighted Merge ------------------
    collab_count = int(len(collaborative_all) * weight_collab)
    content_count = int(len(content_based_all) * weight_content)

    collab_slice = collaborative_all[:collab_count]
    content_slice = content_based_all[:content_count]
    combined_list = collab_slice + content_slice

    final_list = list(dict.fromkeys(combined_list))
    return final_list[:5] if final_list else ["No recommendation available
       "]
```

6 PDF Report Generation

A final PDF report `customer_analysis_report.pdf` summarizes:

- Top categories and products
- Sample spending statistics
- Sample segmentation (Low, Medium, High Spenders)
- Example recommendations for a selected customer

Listing 6: Snippet for generating the PDF report.

```
def generate_pdf_report(
    top_categories, top_products, avg_spending,
    customer_segments, example_customer, example_recommendation
):
    pdf = FPDF()
    pdf.set_auto_page_break(auto=True, margin=15)
    pdf.add_page()

    # Title
    pdf.set_font("Arial", style='B', size=16)
    pdf.cell(200, 10, "Customer Purchase Analysis Report", ln=True, align=
        'C')
    pdf.ln(10)
    ...
    pdf.output(report_file)
    print(f"Report generated: {report_file}")
```

This ensures all insights and methodology are neatly captured in a portable format.

7 Findings and Recommendations

7.1 Key Observations

- **Top Categories & Products**: Reveals best-selling product lines and items.
- **Customer Segmentation**: Helps identify and tailor marketing to Low, Medium, and High Spenders.
- **Hybrid Recommendations**: Balances user-similarity insights with item similarities, emphasizing newer purchases.

7.2 Future Improvements

- Dynamically tune α for faster or slower decay of old purchases.
- Introduce user metadata (location, demographics) to refine segmentation.
- Explore advanced models (matrix factorization, deep learning) for more sophisticated recommendations.

8 Conclusion

This project demonstrates how businesses can leverage an AI-powered pipeline to:

1. Efficiently generate or process purchase data
2. Identify key products and spending behaviors
3. Segment customers for targeted promotions
4. Deliver personalized recommendations that account for changing user preferences over time

By balancing Collaborative and Content-Based strategies, with a time decay for older purchases, the system offers both flexibility and relevance. Future enhancements can incorporate richer data sources and more advanced modeling techniques, ensuring a scalable and adaptive retail intelligence solution.

— **End of Report** —