Task 2 Documentation: Building a Lookalike Model for Customer Similarity

Table of Contents

1. Introduction

In today's competitive business environment, understanding customer behavior is crucial for driving personalized marketing strategies. One such strategy is the **Lookalike Model**, which identifies customers who are similar to a given customer based on their profile and transaction history. This documentation provides a **comprehensive, step-by-step guide** to building a Lookalike Model using Python, focusing on customer and product information. The goal is to create a model that is **accurate, scalable, and interpretable**, while following **Standard Operating Procedures (SOP)**.

---

2. Problem Statement

The task is to build a Lookalike Model that:

- Takes a user's profile and transaction history as input.
- Recommends 3 similar customers based on their profile and transaction history.
- Assigns a similarity score to each recommended customer.
- Outputs the top 3 lookalikes for the first 20 customers in a CSV file (`Lookalike.csv`).

Key Requirements

- Use both customer and product information.
- Ensure the model is accurate and logical.
- Provide meaningful recommendations and similarity scores.

---

3. Solution Overview

The solution involves the following steps:

1. **Data Preparation**: Load and preprocess customer and transaction data.
2. **Feature Engineering**: Create meaningful features from the raw data.
3. **Model Development**: Normalize features, encode categorical variables, and compute similarity scores.
4. **Results**: Generate and save the top 3 lookalikes for each customer.

---

4. Data Preparation

4.1 Data Loading

The first step is to load the datasets:

- `Customers.csv`: Contains customer information such as `CustomerID`, `Region`, and `SignupDate`.
- `Transactions.csv`: Contains transaction details such as `TransactionID`, `CustomerID`, `ProductID`, and `TotalValue`.
- `Products.csv`: Contains product details such as `ProductID`, `ProductName`, and `Category`.

```
import pandas as pd

# Load datasets
customers = pd.read_csv('Customers.csv')
transactions = pd.read_csv('Transactions.csv')
products = pd.read_csv('Products.csv')
```

4.2 Data Cleaning

Ensure the data is clean and ready for analysis:

- **Handle Missing Values**: Check for missing values and decide whether to impute or drop them.
- **Convert Data Types**: Convert `SignupDate` to datetime for easier manipulation.

```
# Check for missing values
print(customers.isnull().sum())
print(transactions.isnull().sum())
print(products.isnull().sum())

# Convert 'SignupDate' to datetime
customers['SignupDate'] = pd.to_datetime(customers['SignupD
ate'])
```

4.3 Feature Engineering

Create customer-level features that capture key aspects of customer behavior:

- **Region**: Customer's location (categorical feature).
- **Tenure**: Number of days since the customer signed up (captures customer loyalty).
- **TotalSpend**: Total amount spent by the customer.
- **TotalQuantity**: Total number of items purchased.
- **TransactionFrequency**: Number of transactions made by the customer.
- **PreferredCategory**: Most frequently purchased product category.

```
# Merge datasets
customer_transactions = pd.merge(customers, transactions, o
n='CustomerID')
customer_transactions = pd.merge(customer_transactions, pro
ducts, on='ProductID')

# Feature engineering
features = customer_transactions.groupby('CustomerID').agg
({
    'Region': 'first',
    'SignupDate': 'first',
    'TotalValue': 'sum',
    'Quantity': 'sum',
    'TransactionID': 'count',
    'Category': lambda x: x.mode()[0]
}).reset_index()

# Calculate tenure
features['Tenure'] = (pd.Timestamp.now() - features['Signup
Date']).dt.days
features = features.drop(columns=['SignupDate'])

# Rename columns
features.columns = ['CustomerID', 'Region', 'TotalSpend',
'TotalQuantity', 'TransactionFrequency', 'PreferredCategor
y', 'Tenure']
```

5. Model Development

5.1 Normalization

Normalize numerical features to ensure they are on the same scale. This is crucial for similarity computation, as features like `TotalSpend` and `TransactionFrequency` may have different ranges.

```
from sklearn.preprocessing import MinMaxScaler

# Normalize numerical features
scaler = MinMaxScaler()
features_normalized = scaler.fit_transform(features[['Total
Spend', 'TotalQuantity', 'TransactionFrequency', 'Tenur
e']])

# Convert normalized features back to a DataFrame
features_normalized = pd.DataFrame(features_normalized, col
umns=['TotalSpend', 'TotalQuantity', 'TransactionFrequenc
y', 'Tenure'])
features_normalized['CustomerID'] = features['CustomerID']
```

5.2 One-Hot Encoding

Encode categorical features (`Region` and `PreferredCategory`) using one-hot encoding. This converts categorical variables into a binary matrix, making them suitable for similarity computation.

```
# One-hot encode categorical features
features_encoded = pd.get_dummies(features, columns=['Regio
n', 'PreferredCategory'])

# Add one-hot encoded features to the normalized DataFrame
for col in features_encoded.columns:
    if col.startswith('Region_') or col.startswith('Preferr
edCategory_'):
        features_normalized[col] = features_encoded[col]
```

5.3 Similarity Computation

Compute similarity scores using **Cosine Similarity**. Cosine Similarity measures the cosine of the angle between two vectors, making it suitable for high-dimensional data.

```
from sklearn.metrics.pairwise import cosine_similarity

# Compute cosine similarity matrix
similarity_matrix = cosine_similarity(features_normalized.d
rop('CustomerID', axis=1))

# Convert similarity matrix to a DataFrame
similarity_df = pd.DataFrame(similarity_matrix, index=featu
res_normalized['CustomerID'], columns=features_normalized
['CustomerID'])
```

6. Results and Output

Generate the top 3 lookalikes for each of the first 20 customers and save the results in `Lookalike.csv`.

```
import numpy as np

# Initialize a dictionary to store lookalike recommendation
s
lookalike_map = {}

# Iterate over the first 20 customers
for i in range(20):
    customer_id = features_normalized.iloc[i]['CustomerID']
    similarity_scores = similarity_matrix[i]

    # Exclude the customer themselves and get the top 3 sim
ilar customers
    top_indices = np.argsort(similarity_scores)[-4:-1]  # E
xclude self and get top 3
    top_customers = [(features_normalized.iloc[idx]['Custom
erID'], similarity_scores[idx]) for idx in top_indices]
```

```
    # Store the results in the dictionary
    lookalike_map[customer_id] = top_customers

# Save results to a CSV file
with open('Lookalike.csv', 'w') as f:
    for cust_id, similar_customers in lookalike_map.items
():
        f.write(f"{cust_id}, {similar_customers}\\n")
```

---

7. Conclusion

The Lookalike Model successfully identifies similar customers based on their profiles and transaction history. By leveraging feature engineering, normalization, and similarity computation, the model provides actionable insights for personalized marketing strategies. The results are saved in `Lookalike.csv`, which can be used for further analysis or integration into marketing workflows.

---

8. Appendix

8.1 Python Packages Used

- **Pandas**: Data manipulation and analysis.
- **NumPy**: Numerical computations.
- **Scikit-learn**: Machine learning tools (e.g., `MinMaxScaler`, `cosine_similarity`).

8.2 Code Walkthrough

- **Data Preparation**: Load and preprocess data.
- **Feature Engineering**: Create meaningful features.
- **Model Development**: Normalize and encode features, compute similarity scores.
- **Results**: Generate and save recommendations.

8.3 FAQs

1. **Why use Cosine Similarity?**

- Cosine Similarity is effective for high-dimensional data and measures the angle between vectors, making it suitable for customer similarity.

2. **How to handle missing data?**
    - Use techniques like imputation or drop missing values based on the dataset.
3. **Can this model be scaled to larger datasets?**
    - Yes, by using distributed computing frameworks like Spark or optimizing the code for performance.

8.4 Advanced Topics

- **Dimensionality Reduction**: Use PCA or t-SNE to reduce the number of features.
- **Clustering**: Apply clustering algorithms (e.g., K-Means) to group similar customers.
- **Advanced Similarity Metrics**: Explore metrics like Jaccard Similarity or Mahalanobis Distance.

8.5 Best Practices

- **Feature Selection**: Choose features that are relevant and meaningful.
- **Model Evaluation**: Use metrics like precision, recall, and F1-score to evaluate the model.
- **Documentation**: Maintain clear and detailed documentation for reproducibility.

8.6 Limitations and Future Work

- **Limitations**:
    - The model assumes that all features are equally important, which may not always be true.
    - It relies on historical data, which may not capture future trends.
- **Future Work**:
    - Incorporate real-time data for dynamic recommendations.

- Use machine learning models (e.g., collaborative filtering) for more accurate predictions.
