Project Report

Dimensionality Reduction and Association Rule Mining in Online Retail Transactions

Ayyub Orujzade

University of Warsaw
Faculty of Economic Sciences

January 2024

Introduction

In this project, I will use dimensionality reduction techniques and association rule mining to explore the intricate dynamics of online retail transactions, uncovering patterns in customer behavior and meaningful relationships between products.

Dimensionality Reduction:

Our approach to dimensionality reduction complies with Quantity, UnitPrice, and InvoiceDate's temporal details. As we embark on the Principal Component Analysis (PCA) journey, we hope to simplify the complexities of online retail transactions by reducing this multidimensional data to its essential components.

Association Rule Mining:

We look at the field of association rule mining, going beyond dimensions in our pursuit to understand the complex relationships that exist between products Cancellations are removed, and transactions are transformed into binary narratives - did a product make its way into a customer's basket or not? Our guide is the Apriori algorithm, which reveals the common item sets that intersperse the history of online retail.

Data overview

The Online Retail dataset provides a thorough view of transactions between customers and a wide range of products, capturing the pulse of an online marketplace. The dataset's entries, which each depict a distinct transaction, highlight the options and exchanges that take place in the online market.

Features:

InvoiceNo: Invoice number, assigned to each transaction.

StockCode*:* Product code, assigned to each distinct product.

Description: Product name.

Quantity: The quantities of each product per transaction.

InvoiceDate: Date and time of each transaction.

Format: Day and time when each transaction was generated.

UnitPrice: Unit price of the product in sterling.

CustomerID: Customer number.

Country: Country where each customer resides.

Dimensionality Reduction

Loading Data

The first step is to load the ***Online Retail dataset***, which will serve as a basis for our exploration. The dataset contains a large number of online retail transactions, each with unique characteristics that provide insights into customer behavior, product interactions, and transactional details.

```
import pandas as pd
df=pd.read_csv('C:/Users/orucz/OneDrive/Рабочий стол/UW
materials/python/Online Retail.csv')
df.head()
```

Here, we use the Pandas library to retrieve the dataset from its online source and make it available for analysis.

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom

Filtering

Dimensionality reduction requires selecting features that capture the essence of the dataset. In this case, we concentrate on two key features: ***Quantity*** (the number of products involved in each transaction) and ***UnitPrice*** (the cost of a product per unit).

```
df_filtered = df[['Quantity', 'UnitPrice']]
df_filtered.head()
```

By focusing on these features, we will reduce the transactional dynamics to a more manageable form.

Standardization

Standardization is a critical preprocessing step that ensures all features have the same scale. This is especially important for techniques like PCA, which are sensitive to the size of the input features.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
standardizedFeatures = scaler.fit_transform(df_filtered)
```

The ***StandardScaler*** from ***scikit-learn*** is used to transform the features into a mean of **0** and a standard deviation of **1**.

Covariance Matrix

The covariance matrix provides information about the relationships between the original features. Each element of the matrix represents the covariance of two features, while the diagonal elements represent the variances of individual features. Here is the code and explanation:

```
import numpy as np
covariance_matrix = np.cov(standardizedFeatures, rowvar=False)
print(covariance_matrix)
```

```
[[ 1.00000185 -0.00123493]
 [-0.00123493  1.00000185]]
```

To calculate the covariance matrix, we use NumPy's ***cov*** function. The ***rowvar=False*** argument indicates that each column represents a variable, making it easier to interpret the matrix.

Eigenvectors and Eigenvalues

The covariance matrix generates eigenvectors and eigenvalues. An ***eigenvector*** is a vector that changes scale rather than direction when multiplied by a matrix. An ***eigenvalue*** is a scalar indicating how much the eigenvector was scaled.

```
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
print("Eigenvalues\n")
print(eigenvalues)
print("Eigenvectors\n")
print(eigenvectors)
```

```
Eigenvalues

[1.00123677 0.99876692]
```

```
Eigenvectors

[[ 0.70710678  0.70710678]
 [-0.70710678  0.70710678]]
```

Sorting Eigenvalues

Eigenvalues are the amount of variance explained by the corresponding eigenvectors. To identify the principal components with the most variance, eigenvalues are typically sorted in descending order.

```
indicesSorted = np.argsort(eigenvalues)[::-1]
eigenvaluesSorted = eigenvalues[indicesSorted]
print(eigenvaluesSorted)
```

```
[1.00123677 0.99876692]
```

PCA (Principal Component Analysis)

Principal Component Analysis is a dimensionality reduction technique that converts the original features into a new set of uncorrelated variables known as principal components. These components capture the most variance from the data, allowing us to represent it in a lower-dimensional space.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
reducedFeatures = pca.fit_transform(standardizedFeatures)
print(reducedFeatures)
```

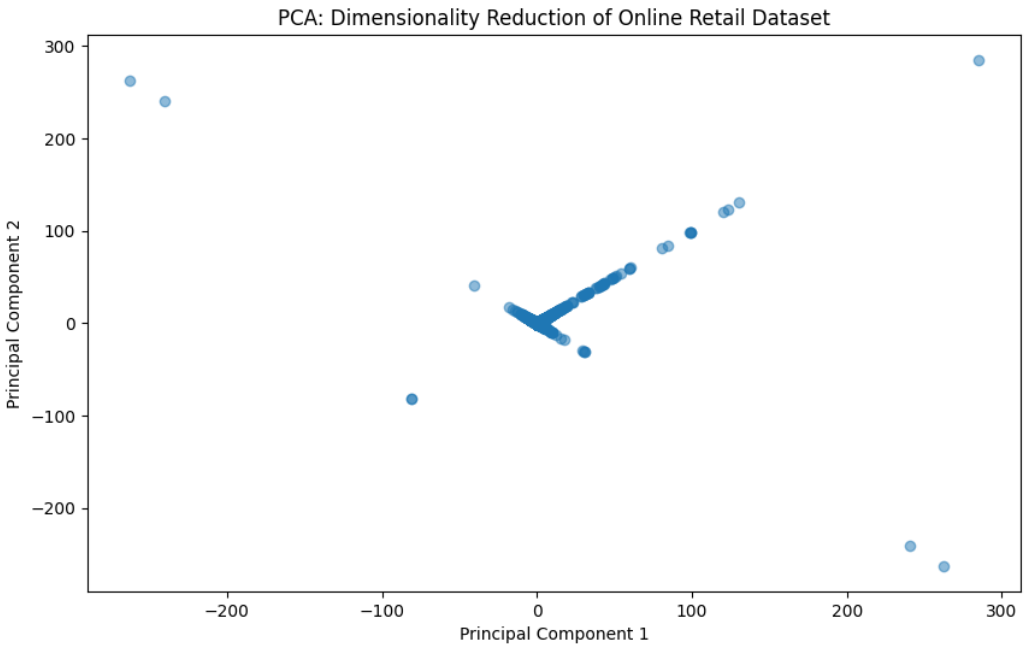
Here, we define an instance of the PCA class and specify that we want to reduce the dimensionality to two components. The fit transform method is then used to complete the transformation.

```
[[-0.0035445 , -0.02658016],
 [ 0.0025941 , -0.02044156],
 [-0.00856774, -0.01863378],
 ...,
 [ 0.01463289, -0.0213724 ],
 [ 0.01463289, -0.0213724 ],
 [ 0.02372158, -0.01876851]])
```

Visualization

Visualization is necessary to gain insights into the data's reduced-dimensional representation. A scatter plot is an effective tool for visualizing the relationships between variables.

```
# Visualize the Scree Plot
plt.figure(figsize=(10, 6))
plt.scatter(reducedFeatures [:, 0], reducedFeatures [:, 1], alpha=0.5)
plt.title('PCA: Dimensionality Reduction of Online Retail Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



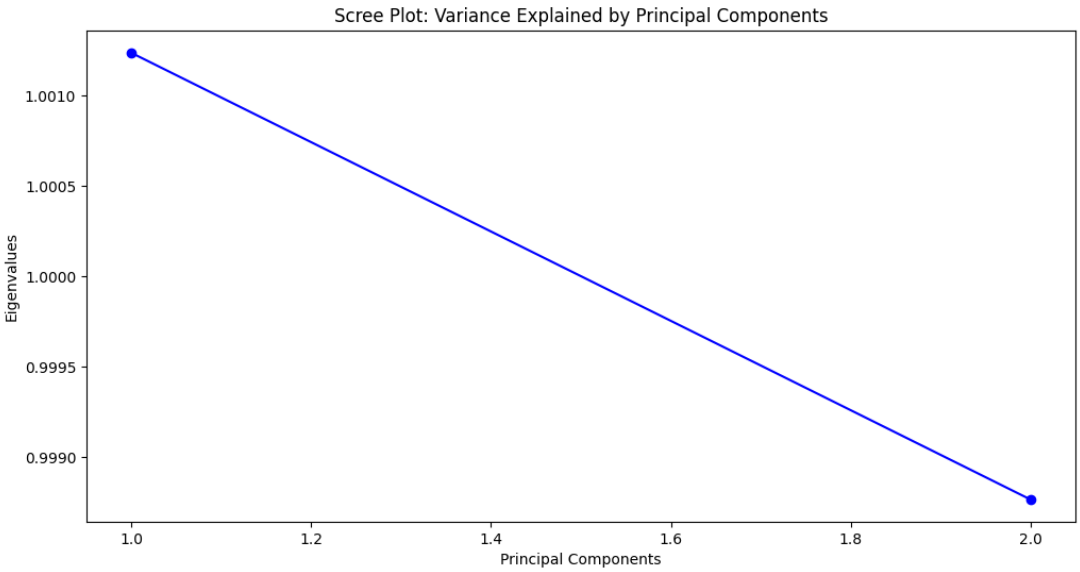
Each point on this scatter plot represents a transaction within the reduced space defined by the principal components. Visual inspection helps to reveal patterns, clusters, and trends in the data.

This comprehensive process allows us to reduce the complexities of the Online Retail dataset to a more manageable size, paving the way for further analysis and interpretation of the reduced-dimensional representation.

Scree Plot

The scree plot graphically represents the explained variance ratio for each principal component. It helps us understand how much of the total variance in the data is captured by each component. Here is the code and explanation:

```
from matplotlib import pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(range(1, len(pca.explained_variance_) + 1),
pca.explained_variance_, marker='o', linestyle='-', color='b')
plt.title('Scree Plot: Variance Explained by Principal Components')
plt.xlabel('Principal Components')
plt.ylabel('Eigenvalues')
plt.show()
```



Determining the Optimal Number of Components

The optimal number of principal components can be determined by looking at the "***elbow***" point in the Scree Plot, where adding more components does not significantly contribute to explaining additional variance.

```
# Identifying the optimal number of components
cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)
# Setting the threshold for cumulative explained variance
threshold = 0.95  # Adjust as needed
# Finding the index where the cumulative explained variance surpasses the
threshold
optimal_components = np.argmax(cumulative_variance_ratio >= threshold) + 1
# Printing the results
print(f"Cumulative Explained Variance Ratio: {cumulative_variance_ratio}")
print(f"Optimal Number of Components to Explain {threshold * 100}%
Variance: {optimal_components}")
```

```
Cumulative Explained Variance Ratio: [0.50061746 1.0]
Optimal Number of Components to Explain 95.0% Variance: 2
```

1. **Cumulative Explained Variance Ratio:**
   - ***pca.explained_variance_ratio_*** provides the ratio of explained variance for each principal component.
   - ***np.cumsum()*** calculates the cumulative sum of these ratios.

2. **Setting the Threshold**:
   - ***threshold*** is set to a chosen value, indicating the desired amount of cumulative explained variance. In this case, it's set to 95%.
3. **Finding the Optimal Number of Components**:
   - The function ***np.argmax(cumulative_variance_ratio >= threshold)*** identifies the index where the cumulative explained variance exceeds or equals the threshold.
   - To count the number of components, we add **+1** since indices start at **0**.

We can determine how much of the original data's variance is preserved in the reduced-dimensional representation by looking at the cumulative explained variance ratio and the ideal number of components. This helps us to balance dimensionality reduction with the preservation of important information when deciding how many components to include in our analysis.

Association Rule

A data mining technique called ***association rule mining*** is used to find intriguing connections or patterns in a dataset. It assists in identifying relationships between various items that frequently occur together in transactions when used in the context of retail or transactional data. Market basket analysis is a popular application where the objective is to find products that are frequently bought together.

Data Preparation for Association Rule Mining

We focus on the ***'InvoiceNo'*** and '***Description***' columns. Each row in this dataset represents a transaction (an invoice), and the '***Description***' column lists the items purchased in that transaction.

```
associationData = df[['InvoiceNo', 'Description']]
```

One-Hot Encoding

The transaction data is converted into a matrix format using ***one-hot encoding***. Each column matches to a ***distinct item*** (product description), and each row represents a ***transaction*** (indexed by 'InvoiceNo'). The corresponding cell in a transaction is marked as ***1*** if a product is present; otherwise, it is marked as ***0***.

```
basket =
pd.get_dummies(associationData['Description']).groupby(associationData[
'InvoiceNo']).max()
```

Apriori Algorithm

The ***Apriori algorithm*** is used to find frequent ***itemsets.*** A parameter that sets the minimum support threshold is - ***min_support***, representing the minimum fraction of transactions that must contain a particular itemset for it to be considered frequent. In this example, we set it to 0.02, meaning we are interested in itemsets that appear in at least 2% of the transactions.

```
from mlxtend.frequent_patterns import apriori
# Application of Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(basket, min_support=0.02, use_colnames=True)
```

Generating Association Rules

The association rules are generated using the frequent ***itemsets*** obtained from Apriori. The metric parameter specifies the rules evaluation metric ("lift" in this case). The **min_threshold** parameter specifies the minimum threshold for the chosen metric.

```
from mlxtend.frequent_patterns import association_rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
```

Displaying the Association Rules

I ran the code below, and it generated a set of association rules that indicate interesting relationships between products in the dataset, allowing us to better understand customer purchasing patterns.

```
print(rules)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(ALARM CLOCK BAKELIKE RED)	(ALARM CLOCK BAKELIKE GREEN)	0.041737	0.038649	0.024942	0.597595	15.462244	0.023329	2.389013	0.976065
1	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE RED)	0.038649	0.041737	0.024942	0.645355	15.462244	0.023329	2.702030	0.972929
2	(CHARLOTTE BAG PINK POLKADOT)	(RED RETROSPOT CHARLOTTE BAG)	0.029344	0.040541	0.020309	0.692105	17.071930	0.019119	3.116193	0.969884
3	(RED RETROSPOT CHARLOTTE BAG)	(CHARLOTTE BAG PINK POLKADOT)	0.040541	0.029344	0.020309	0.500952	17.071930	0.019119	1.945018	0.981203
4	(SPACEBOY LUNCH BOX)	(DOLLY GIRL LUNCH BOX)	0.035058	0.033205	0.020772	0.592511	17.844227	0.019608	2.372568	0.978255
5	(DOLLY GIRL LUNCH BOX)	(SPACEBOY LUNCH BOX)	0.033205	0.035058	0.020772	0.625581	17.844227	0.019608	2.577175	0.976380

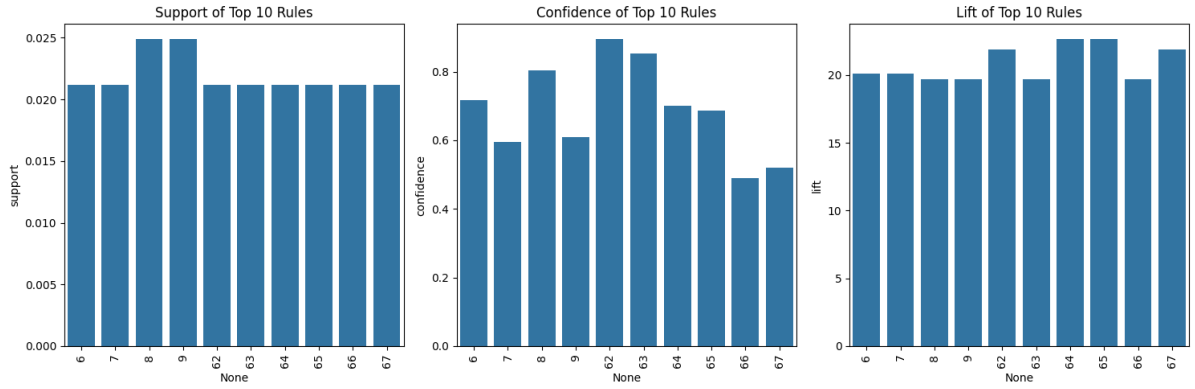
The resulting ***rules*** DataFrame contains details about the discovered association rules, such as ***antecedent*** (premise), ***consequence*** (result), ***support***, ***confidence***, and ***lift***.

Analysis

Let's look at the association rules generated from the Online Retail dataset. We'll look at the ***support***, ***confidence***, and ***lift*** values to gain insight into the patterns we discovered.

```
# Visualization of support, confidence, and lift:
plt.figure(figsize=(15, 5))
# Subplot for Support
plt.subplot(1, 3, 1)
sns.barplot(x=top_rules.index, y=top_rules['support'])
plt.title('Support of Top 10 Rules')
plt.xticks(rotation=90)
# Subplot for Confidence
plt.subplot(1, 3, 2)
sns.barplot(x=top_rules.index, y=top_rules['confidence'])
plt.title('Confidence of Top 10 Rules')
plt.xticks(rotation=90)
# Subplot for Lift
plt.subplot(1, 3, 3)
sns.barplot(x=top_rules.index, y=top_rules['lift'])
plt.title('Lift of Top 10 Rules')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

This code creates ***subplots*** to visualize the support, confidence, and lift values for the top 10 rules.



The support, confidence, and lift values provide insights into the strength and significance of the discovered associations.

Conclusion

This project aimed to explore and analyze the ***Online Retail dataset*** using data science techniques, specifically focusing on ***dimensionality reduction*** through Principal Component Analysis (PCA) and ***association rule mining***.

Dimensionality Reduction (PCA)

The Online Retail dataset featured customer transaction details, including 'Quantity' and 'UnitPrice.'

Summary:

- Employed PCA for dimensionality reduction, achieving a balance between simplicity and information preservation.
- Determined the optimal number of components and visualized feature contributions through Scree Plots.

Association Rule Mining

We used the Online Retail dataset to mine association rules, focusing on 'InvoiceNo' and 'Description.'

Summary:

- Utilized the Apriori algorithm to identify frequent itemsets and generate association rules.
- Analyzed and visualized support, confidence, and lift, revealing valuable insights into product associations.

Overall conclusion

The project demonstrated a thorough examination of the ***Online Retail*** dataset, combining ***association rule mining, PCA,*** and ***dimensionality reduction***, discovered important product associations and insightful patterns in consumer behavior and showed how to use important data science methods to get insightful information from transactional data in the retail industry.

References

1. ***"Online Retail Data Set." (2015).***
   https://archive.ics.uci.edu/dataset/352/online+retail

2. ***Pedregosa, F., et al. (2011). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12, 2825–2830 .***
   https://jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

3. ***McKinney, W. (2010). "Powerful data analysis tools for Python." Proceedings of the 9th Python in Science Conference, 51–56.***
   https://conference.scipy.org/proceedings/scipy2010/mckinney.html

4. ***Hunter, J. D. (2007). "Visualization with Python." Computing in Science & Engineering, 9(3), 90–95.***
   https://ieeexplore.ieee.org/document/4160265

5. ***Han, J., Pei, J., & Yin, Y. (2000). "Mining Frequent Patterns without Candidate Generation." ACM SIGMOD Record, 29(2), 1–12.***
   https://dl.acm.org/doi/10.1145/335191.335372

6. ***Visualization Michael Waskom et al. (2021). "seaborn: Statistical data visualization."***
   https://seaborn.pydata.org/