Apéndice b

Modelación cadena de Markov

Septiembre de 2024

Santiago Mora Cruz

Gabriel Reynoso Escamilla

Paulina Martinez Lopez

Guillermo Villegas Morales

In [ ]:
```
import pandas as pd
import numpy as np
from numpy.linalg import eig, inv
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:
```
df = pd.read_csv('db.csv', usecols = ['fecha', 'material', 'id_cliente', 'ventas'])
df['ventas'] = df['ventas'].astype('float')
df['fecha'] = pd.to_datetime(df['fecha'])

similarity_matrix = pd.read_csv('similarity_matrix.csv', index_col=0)

def monthly_trans_mat_mat(material, df = df):

    # Filter dataframe by material
    df_material = df[df['material'] == material]

    # Determine the global date range across all materials
    material_start_date = df_material['fecha'].min().to_period('M')
    global_end_date = df['fecha'].max().to_period('M')

    # Create a full date range for all months from the global start to end date
    all_months = pd.period_range(start=material_start_date, end=global_end_date, freq='M')

    # Group by year and month, and aggregate 'ventas' per month

    df_material['year_month'] = df_material['fecha'].dt.to_period('M')

    monthly_sales = df_material.groupby('year_month')['ventas'].sum().reset_index()

    # Reindex to include all months in the global date range, filling missing months with 0 ventas
    monthly_sales = monthly_sales.set_index('year_month').reindex(all_months, fill_value=0).reset_index()

    # Initialize activity states: 1 for active, 0 for inactive
    monthly_sales['activity'] = 0

    # Determine activity based on sales
    for i in range(len(monthly_sales)):
        if monthly_sales.loc[i, 'ventas'] > 0:
            monthly_sales.loc[i, 'activity'] = 1
        elif monthly_sales.loc[i, 'ventas'] < 0 and i < len(monthly_sales) - 1:
            # Lookahead: if the next month is positive, mark this month as active
            if monthly_sales.loc[i + 1, 'ventas'] > 0:
                monthly_sales.loc[i, 'activity'] = 1

    # Calculate the transitions
    transitions = monthly_sales['activity'].diff().fillna(0)

    # Initialize the transition matrix
    transition_matrix = np.zeros((2, 2))

    # Count the transitions and fill the transition matrix
    for i in range(1, len(transitions)):
        prev_state = int(monthly_sales['activity'].iloc[i-1])
        current_state = int(monthly_sales['activity'].iloc[i])
        transition_matrix[prev_state, current_state] += 1

    n = transition_matrix[0].sum()
    m = transition_matrix[1].sum()

    transition_matrix[0][0], transition_matrix[0][1] = transition_matrix[0][0]/n, transition_matrix[0][1]/n
    transition_matrix[1][0], transition_matrix[1][1] = transition_matrix[1][0]/m, transition_matrix[1][1]/m

    # Return both the transition matrix and the last active date
    return transition_matrix

def last_date_mat(material, df = df):

    return df[df['material'] == material]['fecha'].max()

def first_date_mat(material, df = df):

    return df[df['material'] == material]['fecha'].min()

def stationary_distr(P):

    p = P[0][1]
    q = P[1][0]

    pi = np.array([q/(p+q), p/(p+q)])

    if P[0][0] == 1 or P[1][1] == 1:
        raise ValueError('Al menos uno de los de la cadena de Markov es absorbente, por lo que la cadena no tiene distribución estacionaria')

    return pi

def t_medio_recurr(pi):

    return 1/pi[1]

def monthly_trans_mat_cli(id_cliente, df = df):

    # Filter dataframe by material
    df_cliente = df[df['id_cliente'] == id_cliente]

    # Determine the global date range across all materials
    client_start_date = df_cliente['fecha'].min().to_period('M')
    global_end_date = df['fecha'].max().to_period('M')

    # Create a full date range for all months from the global start to end date
    all_months = pd.period_range(start=client_start_date, end=global_end_date, freq='M')

    # Group by year and month, and aggregate 'ventas' per month

    df_cliente['year_month'] = df_cliente['fecha'].dt.to_period('M')

    monthly_sales = df_cliente.groupby('year_month')['ventas'].sum().reset_index()

    # Reindex to include all months in the global date range, filling missing months with 0 ventas
    monthly_sales = monthly_sales.set_index('year_month').reindex(all_months, fill_value=0).reset_index()

    # Initialize activity states: 1 for active, 0 for inactive
    monthly_sales['activity'] = 0

    # Determine activity based on sales
    for i in range(len(monthly_sales)):
        if monthly_sales.loc[i, 'ventas'] > 0:
            monthly_sales.loc[i, 'activity'] = 1
        elif monthly_sales.loc[i, 'ventas'] < 0 and i < len(monthly_sales) - 1:
            # Lookahead: if the next month is positive, mark this month as active
            if monthly_sales.loc[i + 1, 'ventas'] > 0:
                monthly_sales.loc[i, 'activity'] = 1

    # Calculate the transitions
    transitions = monthly_sales['activity'].diff().fillna(0)

    # Initialize the transition matrix
    transition_matrix = np.zeros((2, 2))

    # Count the transitions and fill the transition matrix
    for i in range(1, len(transitions)):
        prev_state = int(monthly_sales['activity'].iloc[i-1])
        current_state = int(monthly_sales['activity'].iloc[i])
        transition_matrix[prev_state, current_state] += 1

    n = transition_matrix[0].sum()
    m = transition_matrix[1].sum()

    transition_matrix[0][0], transition_matrix[0][1] = transition_matrix[0][0]/n, transition_matrix[0][1]/n
    transition_matrix[1][0], transition_matrix[1][1] = transition_matrix[1][0]/m, transition_matrix[1][1]/m

    # Return both the transition matrix and the last active date
    return transition_matrix

def last_date_cli(client_id, df = df):

    return df[df['id_cliente'] == client_id]['fecha'].max()

def first_date_cli(client_id, df = df):

    return df[df['id_cliente'] == client_id]['fecha'].min()

def plot_mat(material, df = df):
    df_material = df[df['material'] == material]
    clientes_mas_compran = df_material.groupby('id_cliente')['ventas'].sum().sort_values(ascending=False).head(10)
    order = clientes_mas_compran.index

    fig, _ = plt.subplots()
    graph  = sns.barplot(x = clientes_mas_compran.index, y = list(clientes_mas_compran.values), order = order, palette = 'plasma')

    fig.patch.set_facecolor('#2b2b2b')
    graph.set_facecolor('#1f1f1f')
    graph.tick_params(colors='white')  # Change tick colors
    graph.xaxis.label.set_color('white')  # Change x-axis label color
    graph.yaxis.label.set_color('white')  # Change y-axis label color
    graph.title.set_color('white')

    plt.xticks(rotation = 90)
    title = 'top 10 clientes que compran el material'
    plt.title(title)
    plt.xlabel('id de cliente')
    plt.ylabel('ventas totales')

    return graph

def plot_cli(id_cliente, df = df):
    df_cliente = df[df['id_cliente'] == id_cliente]
    materiales_mas_compran = df_cliente.groupby('material')['ventas'].sum().sort_values(ascending=False).head(10)
    order = materiales_mas_compran.index

    fig, _ = plt.subplots()
    graph  = sns.barplot(x = materiales_mas_compran.index, y = list(materiales_mas_compran.values), order = order, palette = 'plasma')

    fig.patch.set_facecolor('#2b2b2b')
    graph.set_facecolor('#1f1f1f')
    graph.tick_params(colors='white')  # Change tick colors
    graph.xaxis.label.set_color('white')  # Change x-axis label color
    graph.yaxis.label.set_color('white')  # Change y-axis label color
    graph.title.set_color('white')

    plt.xticks(rotation = 90)
    title = 'top 10 materiales que compra el cliente'
    plt.title(title)
    plt.xlabel('material')
    plt.ylabel('ventas totales')


    return graph

def proporcion_negativos_cli(id_cliente, db = df):
    db_cliente = db[db['id_cliente'] == id_cliente]
    return np.round(100*db_cliente[db_cliente['ventas'] < 0].shape[0] / db_cliente.shape[0], 2)

def proporcion_negativos_mat(material, db = df):
    db_material = db[db['material'] == material]
    return np.round(100*db_material[db_material['ventas'] < 0].shape[0] / db_material.shape[0], 2)

# --------------  sistema de recomendaciones ---------------------------------------

def recomendar_productos(id_cliente, num_recomendaciones=10, df = df):
    df_cliente = df[df['id_cliente'] == id_cliente]

    # Obtener los top 5 materiales más comprados por el cliente
    materiales_mas_compran = df_cliente.groupby('material')['ventas'].sum().sort_values(ascending=False).head(10)
    top5_cliente = materiales_mas_compran.index.tolist()

    recomendaciones = pd.Series(dtype=float)
    for material in top5_cliente:
        recomendaciones = recomendaciones.add(similarity_matrix[material], fill_value=0)

    recomendaciones = recomendaciones.groupby(recomendaciones.index).mean().sort_values(ascending=False)
    recomendaciones = recomendaciones[~recomendaciones.index.isin(top5_cliente)]
    return recomendaciones.head(num_recomendaciones)

def recomendar_materiales_similares(material, num_recomendaciones=10):

    # Verificar si el material está en la matriz de similitud
    if material not in similarity_matrix.index:
        raise ValueError(f"El material '{material}' no se encuentra en la matriz de similitud.")

    # Obtener las similitudes de ese material con todos los demás
    similitudes = similarity_matrix[material]

    # Ordenar los materiales por similitud de forma descendente y excluir el propio material
    recomendaciones = similitudes.sort_values(ascending=False).drop(material)

    # Retornar los 10 materiales más similares
    return recomendaciones.head(num_recomendaciones)

clientes = df['id_cliente'].unique()
materiales = df['material'].unique()
```