Autor: Dimitrio Mandamadiotis Kalfagiannis CI 23723158

Tutor: Yaneth Moreno

**DESARROLLO DE UN SISTEMA PARA INFERIR LA DESERCIÓN DE LOS CLIENTES EN UN E-COMMERCE**

**CAPITULO 4**

**Desarrollo, Pruebas y Resultados**

En este capitulo se explican en detalle los procesos que fueron necesarios para implementar el sistema tomando en cuenta la arquitectura planteada en el capitulo anterior, además se muestran los resultados y pruebas realizadas para garantizar que el funcionamiento y rendimiento cumpla los objetivos.

**4.1 Implementación de Módulos**

A continuación se aprecia la implementación de cada uno de los módulos que componen el sistema para inferir la deserción de los clientes en un e-commerce.

**4.1.1 Modelo**

El objetivo de este módulo es generar un modelo en el formato PKL, para esto se decidió crear un programa utilizando la tecnología Jupyter Notebook. El proceso se realizó en fases:

**4.1.1.1 Instalación de dependencias**

La primera fase consiste en importar las dependencias necesarias para realizar el análisis, preprocesamiento y generar el modelo:

import pandas as pd

import datetime

from lifetimes import BetaGeoFitter

from lifetimes.plotting import plot\_frequency\_recency\_matrix

from lifetimes.plotting import plot\_probability\_alive\_matrix

from lifetimes.plotting import plot\_period\_transactions

from lifetimes.plotting import plot\_calibration\_purchases\_vs\_holdout\_purchases

from lifetimes.plotting import plot\_history\_alive

from lifetimes.utils import summary\_data\_from\_transaction\_data

from lifetimes.utils import calibration\_and\_holdout\_data

**4.1.1.2 Cargar el Dataset**

El Dataset que se va a utilizar contiene los datos de transacciones de ventas de una tienda de comercio electrónico con sede en el Reino Unido. Esta tienda de Londres vende regalos y artículos para el hogar y sus clientes provienen de todo el mundo.

El conjunto de datos contiene 500.000 Filas y 8 columnas en las cuales esta representadas las transacciones de ventas de la tienda durante un año. Las columnas son las siguientes:

* **TransactionNo** (categórico): un número único de seis dígitos que define cada transacción. La letra “C” en el código indica una cancelación.
* **Date**(numérico): la fecha en que se generó cada transacción.
* **ProductNo** (categórico): un carácter único de cinco o seis dígitos que se utiliza para identificar un producto específico.
* **Product** (categórico): nombre del producto/artículo.
* **Price** (numérico): el precio de cada producto por unidad en libras esterlinas (£).
* **Quantity** (numérico): la cantidad de cada producto por transacción. Valores negativos relacionados con transacciones canceladas.
* **CustomerNo** (categórico): un número único de cinco dígitos que define a cada cliente.
* **Country** (categórico): nombre del país donde reside el cliente.

Hay un pequeño porcentaje de cancelación de pedidos en el conjunto de datos. La mayoría de estas cancelaciones se debieron a condiciones de falta de existencias en algunos productos. En esta situación, los clientes tienden a cancelar un pedido porque quieren que todos los productos se entreguen de una vez. Los datos se cargan desde el archivo CSV utilizando la biblioteca Pandas.

transactional\_data = pd.read\_csv('../data/Sales Transaction v.4a.csv')

**4.1.1.3 Preprocesamiento de los Datos**

Para poder trabajar correctamente con los datos, es necesario que estos cumplan con los formatos esperados de la biblioteca lifetimes y que los datos sean válidos, para ello se realizaron las siguientes operaciones sobre el dataset:

1. Se removió la información de precio, país, producto y cantidad ya que no es necesaria para el análisis.
2. Las transacciones canceladas se van a considerar en el modelo como una interacción de parte del cliente, por lo tanto no se eliminarán.
3. Varios productos pueden pertenecer a una misma transacción, para representar esto, el dataset contiene entradas con el mismo número de transacción pero con productos diferentes, como solo es necesario saber si se realizó una transacción y no importan los detalles de la misma, se eliminarán las entradas duplicadas.
4. Es necesario transformar los datos en el campo de fecha al formato *datetime* de Python.

El código que realiza este preprocesamiento es el siguiente:

# Operación 1

transactional\_data = transactional\_data.drop(

columns=['Price', 'Quantity', 'Country', 'ProductName', 'ProductNo'])

# Operación 3

transactional\_data = transactional\_data.drop\_duplicates(subset=['TransactionNo'])

# Operación 4

transactional\_data['Date'] = pd.to\_datetime(

transactional\_data['Date'], format='%m/%d/%Y')

**4.1.1.4 Formato de datos RFM**

Los modelos BTYD y especificamente la biblioteca lifetimes reciben como entrada datos en formato RFM, esto quiere decir que debemos llevar los datos transaccionales a una tabla RFM.

Para transformar nuestra data transaccional a data RFM utilizamos la función summary\_data\_from\_transaction\_data de la biblioteca lifetimes.

rfm\_data = summary\_data\_from\_transaction\_data(

transactional\_data, 'CustomerNo', 'Date', observation\_period\_end='2019-12-31')

**4.1.1.5 Generar el modelo**

Finalmente se va crear un modelo BG/NBD con la data RFM que representa el comportamiento de los clientes del dataset, para ello se utiliza la clase BetaGeoFitter de la biblioteca lifetimes.

model = BetaGeoFitter()

model.fit(rfm\_data['frequency'], rfm\_data['recency'], rfm\_data['T'])

**4.1.1.6 Guardar modelo**

La clase BetaGeoFitter de lifetimes posee una rutina que permite guardar el modelo en un formato PKL para poder ser usado posteriormente.

model.save\_model('../api/modelo.pkl')

**4.1.2 API**

Para implementar la API en el lenguaje de programación Python, se utilizó la biblioteca Flask que abstrae toda la funcionalidad relacionada con programar un servidor web. Esta biblioteca se encarga de implementar todas las reglas del protocolo HTTP.

El objetivo de la API es crear una ruta HTTP que exponga el recurso conditional-probability-alive, el cual permite calcular la probabilidad de vida de uno o mas clientes. El código del servidor web es el siguiente:

# Dependencias

import datetime

import pandas as pd

from flask import Flask, request

from flask\_cors import CORS

from lifetimes import BetaGeoFitter

from lifetimes.utils import summary\_data\_from\_transaction\_data

from lifetimes.utils import calculate\_alive\_path

# Cargar Modelo

model = BetaGeoFitter()

model.load\_model("./modelo.pkl")

app = Flask(\_\_name\_\_)

CORS(app)

# Ruta para verificar la salud del servidor

@app.route("/")

def index():

return "Index Page"

# Recurso

@app.route("/conditional-probability-alive", methods=["POST"])

def conditionalProbabilityAlive():

if request.method == "POST":

customer\_field = "customer"

date\_field = "date"

data = request.json["transactions"]

# Validaciones

if data.get("customer") is None or data.get("date") is None:

return "Bad Request", 400

if type(data.get("customer")) not in (tuple, list):

return "Bad Request", 400

if type(data.get("date")) not in (tuple, list):

return "Bad Request", 400

if len(data.get("date")) != len(data.get("customer")):

return "Bad Request", 400

df = pd.DataFrame.from\_dict(data)

rfm\_data = summary\_data\_from\_transaction\_data(df, customer\_field, date\_field)

rfm\_data["p\_alive"] = model.conditional\_probability\_alive(

rfm\_data["frequency"], rfm\_data["recency"], rfm\_data["T"]

)

result = rfm\_data.to\_dict("index")

for i, \_row in rfm\_data.iterrows():

path = (

calculate\_alive\_path(

model, df[df[customer\_field] == i], date\_field, 500

)

.to\_numpy()

.tolist()

)

result[i]["path"] = [item for sublist in path for item in sublist]

return {"result": result}

else:

raise RuntimeError("Unsupported method {}".format(request.method))

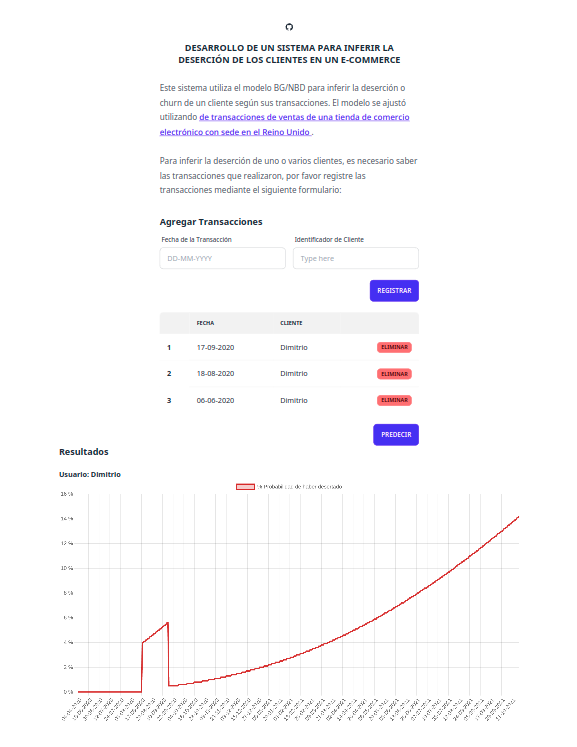
Al inicio, el programa importa las dependencias necesarias para su ejecución: Pandas, datetime, lifetimes y Flask; luego de esto, se inicializa y carga el modelo, acto seguido, se implementa una ruta de inicio para verificar que el servidor se está ejecutando y finalmente se implementa el recurso objetivo.

El recurso /conditional-probability-alive recibe una lista de transacciones de clientes, transforma esa información en una tabla RFM que es utilizada como entrada al modelo para calcular el “camino de vida” o probabilidad de que cada cliente esté activo durante 500 días luego de la primera transacción.

El valor de 500 días se puede parametrizar, pero se decidió mantenerlo constante para evitar una sobrecarga del servidor en caso de que se solicite procesar mucha información. Finalmente el recurso retorna toda la información que ha calculado, incluyendo la tabla RFM.

**4.1.3 Aplicación Web**

Para construir la interfaz web se usó Vue.js como framework y Nuxt como metaframework para ahorrar tiempo de desarrollo. Vue.js se encarga de abstraer y gestionar todo el dinamismo, mientras que Nuxt provee de una estructura al proyecto y abstrae toda la lógica de compilación, optimización y manejo de dependencias del proyecto.

****

La aplicación esta compuesta de una sola vista llamada “index” donde se representan todos los aspectos de la interfaz.

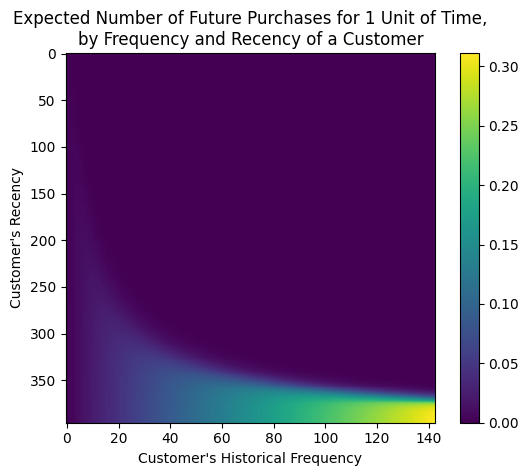
**4.2 Pruebas de Módulos**

**4.2.1 Modelo**

Para validar el modelo se realizaron varias pruebas y análisis de la información obtenida, estas fueron las conclusiones obtenidas.

**4.2.1.1 Matriz de Frecuencia, Recencia y Compras Esperadas**

Inicialmente lo que buscamos con el modelo es predecir el número esperado de transacciones que un cliente realizará el próximo periodo de tiempo, es por esto que podemos validar la calidad del funcionamiento de nuestro modelo graficando la relación que existe entre la frecuencia, la recencia y el número esperado de compras.

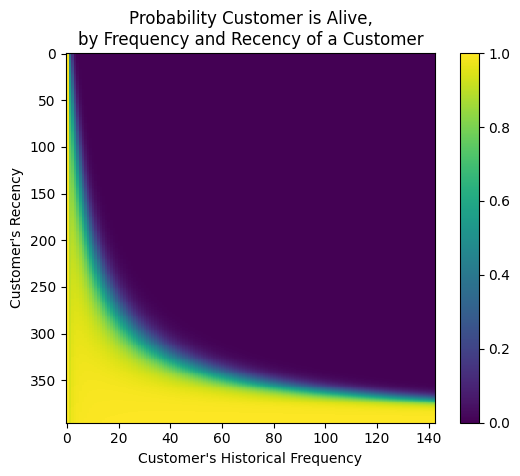


Podemos ver que si un cliente ha interactuado más de 120 veces, y su última interacción la realizó luego de 350 días de estar activo, entonces podemos considerarlo como uno de los mejores clientes (abajo a la derecha). Los clientes menos activos son los que están en la esquina superior derecha: compraron mucho rápido y no lo han vuelto a hacer en semanas.

También está la “cola” alrededor de las coordenadas (20, 300). Eso representa al cliente que compra con poca frecuencia, pero que se ha visto recientemente, por lo que podría volver a comprar; no se tiene seguridad de si ha dejado de ser cliente o solo está en un periodo entre compras.

**4.2.1.2 Matriz de Frecuencia, Recencia y Probabilidad de vida**

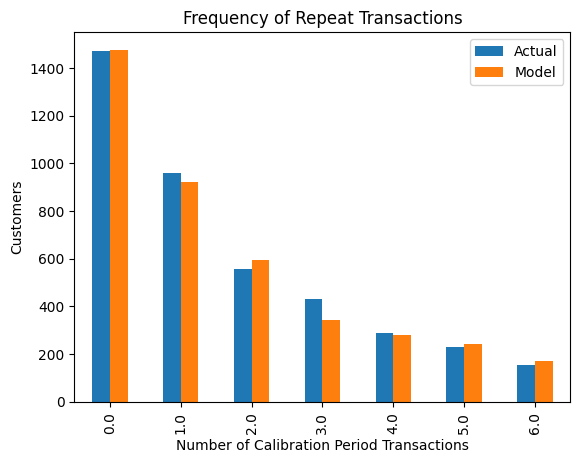
Otra información interesante a evaluar es la relación entre la frecuencia, recencia y la probabilidad de vida de un cliente.

****

De forma similar a la matriz anterior, si los clientes tienen una frecuencia alta de compra y los hemos visto durante tiempos prolongados, su probabilidad de ser clientes es muy alta (abajo y a la derecha). De igual manera si un cliente se ha visto recientemente tiene una alta probabilidad de estar activo (arriba a la izquierda). Esta información que nos da el modelo tiene sentido y se puede asociar con el comportamiento de los clientes en el mundo real.

**4.2.1.3 Evaluación del ajuste del modelo**

En el siguiente gráfico se evalúan los parámetros de ajuste del modelo, para ello se comparan los datos que se utilizaron para ajustar el modelo con datos artificiales simulados con los parámetros del modelo ajustado.



Se puede observar que los datos reales y los datos simulados se alinean bien, esto prueba que el modelo es bueno y representa muy bien el comportamiento de los clientes.

**4.2.1.4 Validación Cruzada**

Podemos dividir el conjunto de datos en un grupo de datos de período de calibración y otro grupo de datos de reserva. Esto es importante ya que se quiere probar cómo funciona el modelo en datos que aún no se han visto. La biblioteca lifetimes tiene una función para particionar el conjunto de datos:

summary\_cal\_holdout = calibration\_and\_holdout\_data(transactional\_data,

'CustomerNo', 'Date',

calibration\_period\_end='2019-06-09',

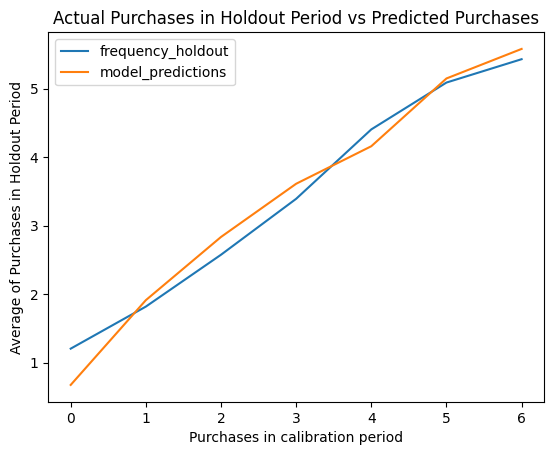
observation\_period\_end='2019-12-09')

Ahora que se particionó la data, se procede a ajustar el modelo con los datos del periodo de calibración:

model.fit(summary\_cal\_holdout['frequency\_cal'],

summary\_cal\_holdout['recency\_cal'], summary\_cal\_holdout['T\_cal'])

Finalmente se compara las compras del periodo de reserva, con las compras esperadas por el modelo en ese mismo periodo:

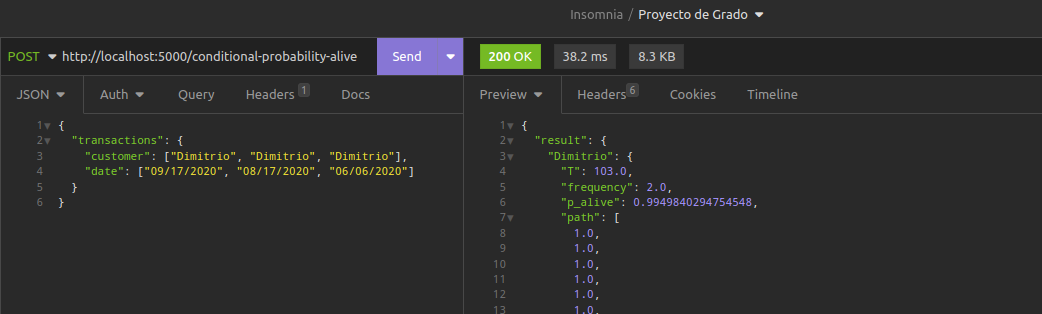


Se puede apreciar una gran similitud entre las compras reales del periodo donde el modelo no tiene conocimiento y las compras que el mismo predijo, viendo esto y las pruebas anteriores podemos concluir que el modelo tiene un rendimiento suficientemente bueno para ser usado en el sistema.

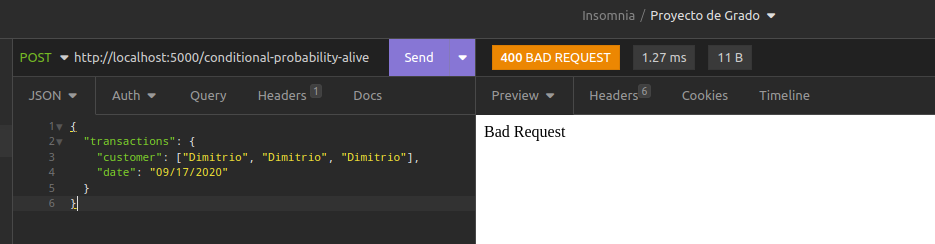
**4.2.2 API**

Para probar que la API cumple los requerimientos, se utilizó un software para hacer peticiones HTTP llamado Insomnia, con el se realizaron varias llamadas al recurso /conditional-probability-alive con diferentes parámetros de entrada y así verificar que la respuesta cumple con los objetivos.

En el primer caso enviamos una petición válida con información sobre varias transacciones y el recurso responde correctamente a la solicitud.

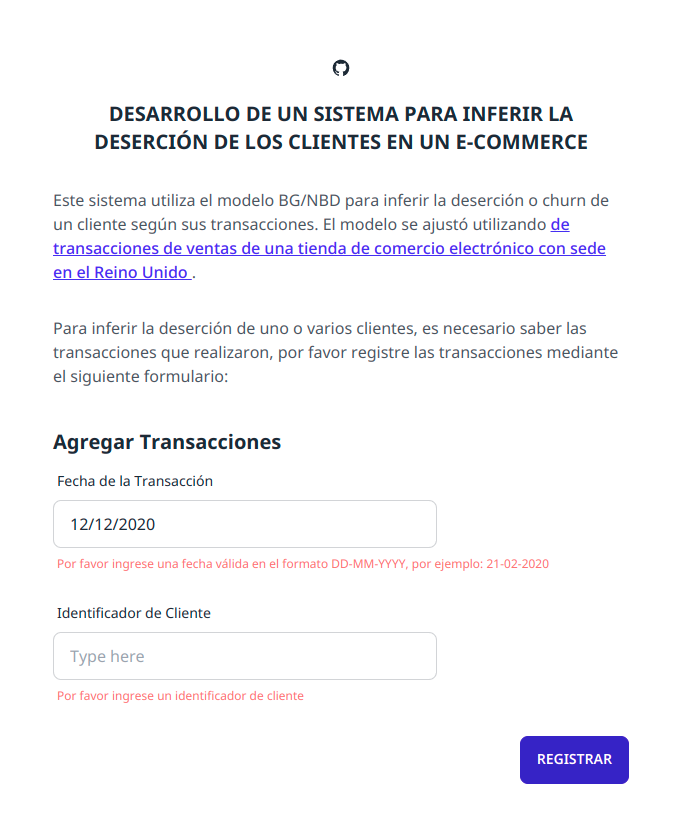


En el segundo caso hacemos una petición inválida a la API y esta responde con el mensaje adecuado.



**4.2.3 Aplicación Web**

Como el listado de transacciones debe tener un formato especifico, se valida la entrada del usuario para garantizar esta estructura, para ello se muestra un texto de ayuda y mensajes con información sobre los diferentes errores de entrada posibles.



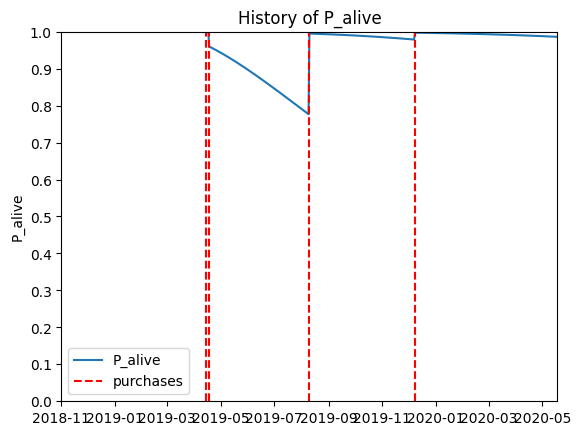
**4.3 Resultados**

**4.3.1 Modelo**

A continuación se van a presentar ejemplos de clientes dentro del dataset, sus resultados y un análisis de la información obtenida:

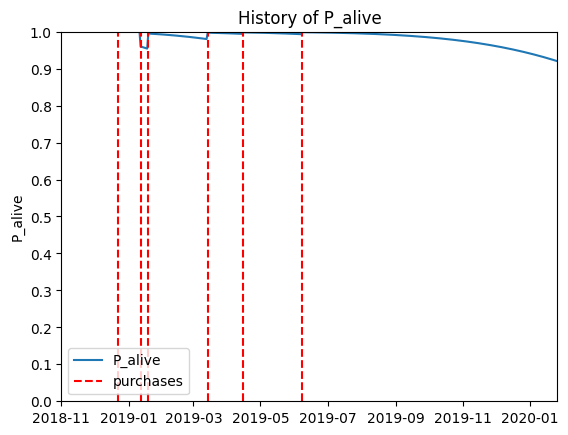
**4.3.1.1 Ejemplo 1**

En este ejemplo se aprecia el comportamiento de un cliente que realizó dos compras consecutivas, al inicio la probabilidad de ser considerado un cliente baja con una mayor rapidez, luego de que el cliente realiza otra compra se puede ver que la probabilidad ahora disminuye más lentamente. Finalmente luego de la cuarta compra se puede observar que este cliente tiene una relación saludable con la tienda y tiene un patrón de compra definido.



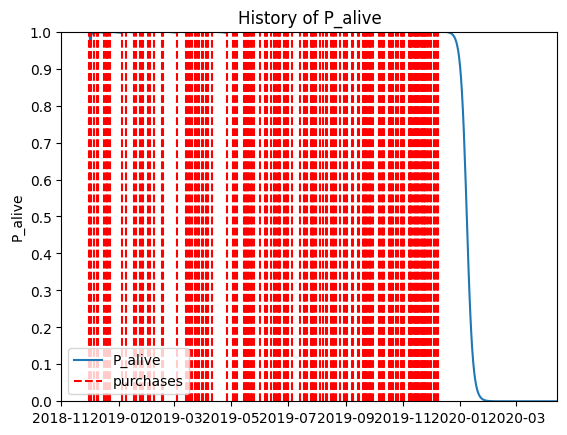
**4.3.1.3 Ejemplo 2**

Este cliente tiene un comportamiento similar al ejemplo anterior, se puede ver que realizó más compras durante la primera mitad del año pero luego no compró mas por el resto del año, esto ocasiona que su probabilidad de ser un cliente activo baje lentamente. A pesar de que el usuario no comprara más durante la segunda mitad del año, no se puede descartar tan rápidamente ya que su patrón de compras es ligeramente extendido.



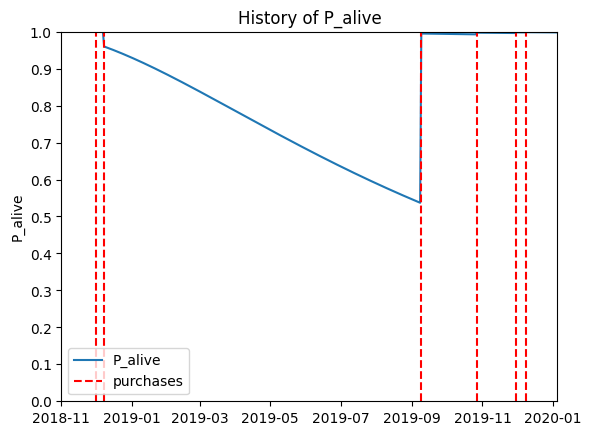
**4.3.1.3 Ejemplo 3**

Este caso es de interés ya que se puede apreciar a un cliente que compra muy seguido durante todo el año, luego de pasar unos días sin comprar, el modelo penaliza fuertemente su probabilidad de seguir siendo cliente ya que no mantiene su comportamiento anterior.



**4.3.1.4 Ejemplo 4**

En este ejemplo el cliente realizó dos compras y durante el año la probabilidad de vida del cliente fue disminuyendo pero no fue descartado, como vemos luego el cliente regresó y se completó su patrón de compra.



**4.3.2 API y Aplicación Web**

A continuación podemos observar los resultados obtenidos al utilizar el sistema para predecir la probabilidad de deserción de un cliente dada una lista de transacciones del mismo.

Esta prueba verifica que la integración entre la aplicación web y la API funciona correctamente y que el sistema está cumpliendo con la funcionalidad esperada.

