Manejo de Inventario

David Peña Peralta

2025-07-11

Table of contents

1 Introducción

Dentro del área del Control Estocástico, una de los problemas más conocidos son los problemas de inventario. Donde se presenta una bodega con capacidad máxima K. Cada etapa se extrae una cantidad de mercancía, la que denotaremos como la demanda D_t, y se solicita una cantidad del producto a_t, obteniendo finalmente el nivel de inventario X_t (ver Sutton and Barto (2018)). En general se busca minimizar los costos de la bodega (costos por almacenamiento, costos por pérdida, entre otros).

2 Formulación del Proceso de Decisión de Markov.

Para nuestro problema consideraremos un supermercado, centrado en uno de sus pasillos. Suponiendo que en un pasillo se almacena un solo tipo de producto. Definiremos a K la cantidad máxima de producto en el pasillo, X_t a la cantidad del producto disponible para la venta (o la cantidad de producto en el pasillo). Nuestra demanda, o producto solicitado, será denotado por D_t y se considerará una colección de v.a i.i.d. Finalmente, la cantidad recolocada en el pasillo, o producto pedido, será denotada por a_t. Entonces, nuestro conjunto de estados $\mathcal{S}$ está dado por el siguiente conjunto.

$$\mathcal{S} = \{s \in \mathbb{Z}^+ : 0 \leq s \leq K\}. \tag{2.1}$$

Nuestro conjunto de acciones $\mathcal{A} = \mathcal{S} = \mathbb{Z}^+$, y para $x \in \mathcal{S}$ nuestro conjunto de acciones admisibles esta dado por

$$\mathcal{A}(x) = \{a \in \mathcal{A} : 0 \leq a \leq K - x\}.$$

3 Dinámica del Modelo.

Recordando la fórmula para nuestro modelo.

$$X_{t+1} = f(X_t, a_t), f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}.$$

Entonces el modelo que usaremos esta dado por

$$X_{t+1} = (X_t + a_t - \eta X_t - D_{t+1})^+, \tag{3.1}$$

donde a_t es la cantidad de producto recolodado al final del dia t, η es el factor descomposición, D_t es la demanda del prodcucto en la dia t y $(\cdot)^+ = \max\{\cdot, 0\}$.

4 Descripción y Justificación del Modelo.

El modelo Equation 3.1 pretende responder a la pregunta que denota el modelo ¿Cuánto producto tendré disponible al dia siguiente?. Lo anterior menciona que nuestras etapas $t \in \mathcal{T} = \{t \in \mathbb{Z}^+ : t \leq T, T \in \mathbb{N}\}$ representaran los dias dentro de un periodo T, t hace referencia al dia actual, y $t+1$ al dia siguiente. Entonces el modelo general esta dado por

$$X_{t+1} = (\text{Today} + \text{In}_t - \text{Out}_t)^+.$$

Esto es, la parte positiva del producto que hay "hoy", es decir, X_t. A eso le agregaremos el producto que entrará hoy al final del dia, en nuestro modelo solo habrá ingreso de producto mediante solicitud (En este caso no consideramos un almacenimiento dentro del supermercado), entonces In_t esta dado por nuestras acciones $\text{In}_t = a_t$.

La parte que saldrá consta de dos elementos. En general consideramos la cantidad de producto que se compró en el dia t. Sin embargo, desconocemos la cantidad requerida, haciendo referencia al dia siguiente. Por lo tanto la demanda está representada por D_{t+1}, la cantidad de producto requerida al dia siguiente. En nuestro modelo también consideramos la salida de producto por considerarse producto no apto para la venta. Entonces

$$\text{Out}_t = D_t + N_t(X_t).$$

Bajo de la suposición que todos los productos poseen el mismo tiempo de vida con periodos de vida distintos supondremos que cada dia, al final, se retira un factor con respecto a la cantidad actual de producto.

$$N_t = \eta X_t$$

$$\text{Out}_t = D_t + \eta X_t$$

Finalmente, nos queda definir la función de costo, en nuestro modelo será la ganancia. Al considerar un periodo finito tenemos que la ganancia total G esta dada por

$$G(x_0, \pi) = \sum_{t=0}^{T} R_t, X_0 = x_0, X_{t+1} = f(X_t, a_t), R_t = R(x_t, a_t, \xi_t)$$

donde π es una politica, $\pi = (a_0, a_1, \dots, a_{N-1})$. y R_t es la ganancia por etapa, en nuestro caso

$$R(x, a, \xi) = P_V \min\{x + a, \xi\} - P_C a - P_C \min \xi - x - a, 0.$$

Al intervenir una variable aleatoria, entonces nos interesaría el valor esperado.

$$\mathcal{G}(x) = E_\pi[G \mid X_t = x].$$

5 Justificación de las acciones.

Ya comentamos que nuestras acciones, serán la cantidad de producto que vamos a solicitar. Entonces nuestras acciones serán números enteros las acciones serán ejecutadas de forma instantea. El conjunto de acciones está dado por

$$\mathcal{A} = \{z \in \mathbb{Z}^+ : z \leq K\}.$$

y para cada $x \in \mathcal{S}$, obtenemos el conjunto de acciones admisibles.

$$\mathcal{A}(x) = \{z \in \mathbb{Z}^+ : z \leq K - x\}.$$

6 Implementación.

Finalmente, mediante la ecuación de Bellman.

$$v(x) = \min_{a \in \mathcal{A}(x)} \{E\left[R(x, a, \xi) + \beta v(f(x, a, \xi))\right]\} \tag{6.1}$$

y las siguientes condiciones iniciales.

- X_0 = 60.
- P_V = 30, P_C = 60.
- eta = 0.1.

y considerando una demanda ξ_t con $\xi_t \sim \text{Geo}(p)$, con $p = 0.05$. Ejecutamos el siguiente código para resolverlo.

```
import numpy as np
import pandas as pd
from scipy.interpolate import CubicSpline


RV_P = 0.05  # Parametro Demanda Aleatoria

REW_SALE = 20  # Precio de Venta
REW_COST = 10  # Costo Unitario
DYN_ETA = 0.1  # Factor de Pérdida
M_K = 100  # Capacidad máxima del inventario

X_0 = 60  # Inventario Inicial
PERIOD = 100 # Periodo Inicial


def demmand(p):
    """
        Define the Random demmand in the inventory model.
        Params:
            p (float): Probabiliy parameter.
```

```
        Return:
            d (float): Random value
    """
    d = np.random.geometric(p)
    return d


def admisible_actions(x, k):
    """
        Calculate the admisible actions for each state X
        Params:
            x (int): state value.
            k (int): max inventory capacity.
        Returns:
            v (list): admisible action set by value x
    """
    v = list(range(k-x+1))
    return v


def reward_function(x, a, d, p_v, p_c):
    """
        Reward Function: Alternative Cost Function
        Params:
            x (int): state value
            a (int): action value
            d (int): demmand value
            p_v (float): sale price
            p_c (float): unitary cost
        Return:
            r (float): Reward by x,a and d with parameters p_v and p_c
    """
    lost_state = 1  # Activate or Deactivate additional cost.
    lost_cost = max(d - x - a, 0)
    c = a + lost_state * lost_cost
    r = p_v * min(x + a, d) - p_c * c
    return r


def dynamic(x, a, d, eta):
    """
        Calcuate the next state
```

```
        Params:
            x (int): state value
            a (int): action value
            d (int): demmand value
            eta (float): loss parameter
        Return:
            xp1 (int): next state
    """
    xp1 = x + a - d - np.floor(eta * x)
    xp1 = max(xp1, 0)
    return xp1

def expected_reward(x, a, d_v):
    """
        Calculate the Expected Reward
        x (int): state value
        a (int): action value
        d_v (list): random demmand vector
        Return:
            data (list) reward demmand function (next to calculate mean)
    """
    p_v = REW_SALE
    p_c = REW_COST
    data = [reward_function(x, a, d, p_v, p_c) for d in d_v]
    return data


def expected_vf(vf, x_0, a, d_v):
    """
        Calculate the function v
        Params:
            vf (function): Interpolation
            x_0 (int): state value
            a (int) : action value
            d_v (list): demmand random sample
        Return:
        vf_d (int): Expected vf function
    """
    vf_d = [float(vf(dynamic(x_0, a, d, DYN_ETA))) for d in d_v]
    vf_d = int(np.array(vf_d).mean())
    return vf_d
```

```
STATES_SET = list(range(M_K + 1))
def evol_vk(vk, hk, beta, der):
        """
        Calculus of v_k
        Params:
            vk (list): iterative function vk
            hk (list): iterative function hk
            beta (float):
            der (list):
        Return:
            vk (list): next function iterative hk
            hk (list): next function iterative vk
    """
    for u, x_k in enumerate(STATES_SET):
        a_x = admisible_actions(x_k, M_K)
    #     print("Admisible Actions",a_x)
        cost_ax = []
        expec_ax = []
        v_function = CubicSpline(STATES_SET, vk)
        for action in a_x:
            cost = [reward_function(x_k, action, d, REW_SALE, REW_COST) for d in der]
            cost = int(np.array(cost).mean())
            cost_ax.append(cost)
            e_vf = expected_vf(v_function, x_k, action, der)
            expec_ax.append(e_vf)
        v_cost = np.array(cost_ax)
        v_vf = np.array(expec_ax)
        v_sum = v_cost + beta * v_vf
        v_max = v_sum.max()
        vk[u] = v_max
        hk[u] = v_sum.argmax()
    return vk, hk


final_df = pd.DataFrame()

for ks in range(10):
    print(ks)
    demmand_vector = [demmand(RV_P) for _ in range(200)]
    v_0 = list(STATES_SET)  # v_0(x) = x
    h_0 = np.zeros(len(v_0))
```

```
    v_mem = [v_0]
    h_mem = [h_0]

    BPAR= 0.9
    for kt in range(25):
        v_1, h_1 = evol_vk(v_0, h_0, beta= BPAR, der = demmand_vector)
        v_mem.append(v_1)
        h_mem.append(h_1)
#       print(f"{kt+1} - {kt}",np.array(v_1) - np.array(v_mem[kt]))
        v_0 = v_1.copy()
        h_0 = h_1.copy()

    test_eval = CubicSpline(STATES_SET, h_mem[-1])

    df = pd.DataFrame(index = list(range(PERIOD)),
                      columns=["state","action","demmand","r"])

    x_hist = [X_0]
    a_hist = []
    d_hist = []
    for q in range(PERIOD):
        a = int(test_eval(x_hist[q]))
        d = demmand(RV_P)
        r = reward_function(x_hist[q], a, d, REW_SALE, REW_COST)
        df.loc[q] = [x_hist[q], a, d, r]
        xp1 = dynamic(x_hist[q], a, d, DYN_ETA)
        x_hist.append(xp1)
        a_hist.append(a)
        d_hist.append(d)
    df['SAMPLE'] = ks
    if ks == 0:
        final_df = df
    else:
        final_df = pd.concat([final_df, df])


# Save data in DataFrame
final_df.to_excel(f"S_{PERIOD}_({M_K, DYN_ETA})_{RV_P}_({REW_SALE, REW_COST}).xlsx")
```

Código para la Visualización.

```python
# Librerias Requeridas
import pandas as pd
import matplotlib.pyplot as plt

# Leemos los datoa generados.
data = pd.read_excel("S_100_((100, 0.1))_0.05_((20, 10)).xlsx", index_col=0)
print(data)

# Hacemos las graficas solcitadas.
samples = data['SAMPLE'].unique()
figure_x, ax0 = plt.subplots(figsize = (6, 4))  # Grafica de Nivel de Inventario.
figure_r, ax1 = plt.subplots(figsize = (6, 4))  # Grafica de la Recompensa por etapa
figure_acr, ax2 = plt.subplots(figsize = (6, 4))  # Gráfica de la Recompensa acumulada.
figure_act, ax3 = plt.subplots(figsize = (6, 4))  # Gráfica de las politicas.
x = 0
r = 0
acr = 0
for w, s in enumerate(samples[:3]):
    data_s = data[data['SAMPLE'] == s]
    ax0.plot(data_s['state'], lw = 0.7, label = f"l_{s}")
    ax0.set_xlabel("Stages")
    ax0.set_ylabel("Level")
    ax1.plot(data_s['r'], lw = 0.7, label = f"l_{s}")
    ax1.set_ylabel("Reward")
    ax1.set_xlabel("Stages")
    ax2.plot(data_s['r'].cumsum(), lw = 0.7, label = f"l_{s}")
    ax2.set_xlabel("Stages")
    ax2.set_ylabel("Acc. Reward")
    ax3.plot(data_s['action'], lw = 0.7, label = f"l_{s}")
    ax3.set_xlabel("Stages")
    ax3.set_ylabel("Action")
    if w == 0:
        x = data_s['state']
        r = data_s['r']
        acr = data_s['r'].cumsum()
    else:
        x += data_s['state']
        r += data_s['r']
        acr += data_s['r'].cumsum()
ax0.plot(x / 3, lw = 0.7, color = "black", label = "x mean")
ax1.plot(r / 3, lw = 0.7, color = "black", label = "r mean")
```

```
ax2.plot(acr / 3, lw = 0.7, color = "black", label = "acc. r mean")
figure_x.legend()
figure_x.tight_layout()
figure_x.savefig("Data_x.png")
figure_r.legend()
figure_r.tight_layout()
figure_r.savefig("Data_r.png")
figure_acr.legend(loc = "upper center")
figure_acr.tight_layout()
figure_acr.savefig("Data_acr.png")
figure_act.legend()
figure_act.tight_layout()
figure_act.savefig("Data_act.png")
plt.show()
```

References

Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: A Bradford Book.