Risk-Averse Airline Revenue Management with Coherent Measures of Risk

Recep Yusuf Bekci

Outline

Revenue Management

Definition

Revenue management is the application of disciplined analytics that predict consumer behaviour at the micro-market level and optimize product availability and price to maximize revenue growth.

The primary aim of revenue management is selling the right product to the right customer at the right time for the right price.

Risk Management

Definition

Risk: A situation involving exposure to danger

For a random variable,

- If it represents cost, risk measures care for higher values
- If it represents revenue, risk measures care for lower values

Risk Measures

Definition

Coherence: a risk measure satisfying the four axioms of translation invariance, subadditivity, positive homogeneity, and monotonicity, is called coherent.

For example, CVaR or Mean Semi-Deviation

Risk Measures

First-Order Mean Semi-Deviation

For a random variable X that represents cost,

$$\rho(X) = \mathbb{E}[X] + \zeta \mathbb{E}[X - \mathbb{E}[X]]_+$$

We are working on revenue, namely reward,

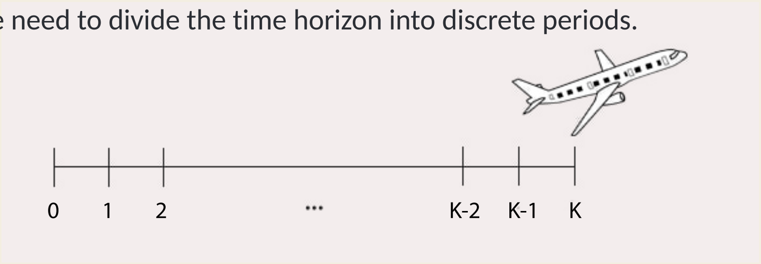
$$\rho(X) = \mathbb{E}[X] - \zeta \mathbb{E}[\mathbb{E}[X] - X]_+$$

where $0 \leq \zeta \leq 1$

Risk Neutral Model

Aydın et al. (2013)

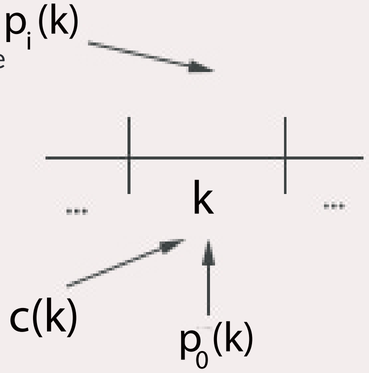
We need to divide the time horizon into discrete periods.



Risk Neutral Model

In each period, there can be three events:

- $p_i(k)$ is the probability of a request for a seat in class i.
- $c(k)$ is the probability of cancellation.
- $p_0(k)$ is the probability of a null event.



Risk Neutral Model

If a request is accepted for class i at a stage, $r_i \geq 0$ is the amount of earned revenue.

We have

$$\sum_{i=1}^{m} p_i(k) + p_0(k) = 1 \quad \textit{for all } k \geq 1.$$

Risk Neutral Model

We also have no-shows. Each customer has a probability β^s of show-up at the departure time. n is the state.

Definition

θ : Overbooking cost.

We have total capacity as C

Definition

κ : Cancellation cost

Risk Neutral Model

Definition

$J_k(n)$ is the maximum expected revenue over periods 1 to k.

At period K,

$$J_K(n) = -\kappa n c(k) - \theta \mathbb{E}\left[B(n, b^s(1 - c(k))) - C\right]_+$$

We have recursive function as

$$J_k(n) = -\kappa n c(k) + p_0(k) J_{k+1}(B(n, 1 - c(k))) +$$
$$\sum_{n=1}^{m} p_i(k) \mathbb{E}\Big[max\big(r(i) + J_{k+1}(B(n, 1 - c(k)) + 1),$$
$$J_{k+1}(B(n, 1 - c(k)))\big)\Big]$$

Risk Averse Model

At period K,

$$V_K(n) = \mu - \zeta * \mathbb{E}\big[\mu + \kappa(n-t) + \theta[s-C]_+\big]_+$$

where $0 \leq \zeta \leq 1$, t is the number of customers who did not cancel, s is the number of customers who show-up and

$$\mu = -\kappa n c(k) - \theta\mathbb{E}\big[B(n, \beta^s(1-c(k))) - C\big]_+$$

Risk Averse Model

We have recursive function as

$$V_k(n) = \mu - \zeta * \mathbb{E}\Big[\sum_{i=0}^{m} p_i(k)\Big(\mu + \kappa(n-t) -$$

$$max(r(i) + V_{k+1}(t+1), V_{k+1}(t))\Big)\Big]_+$$

where $0 \leq \zeta \leq 1$, t is the number of customers who did not cancel and

$$\mu = -\kappa n c(k) + p_0(k) V_{k+1}(B(n, 1-c(k))) +$$

$$\sum_{i=1}^{m} p_i(k)\mathbb{E}\Big[max\Big(r(i) + V_{k+1}(B(n, 1-c(k)) + 1),$$

$$V_{k+1}(B(n, 1-c(k)))\Big)\Big]$$

Computational Experiments

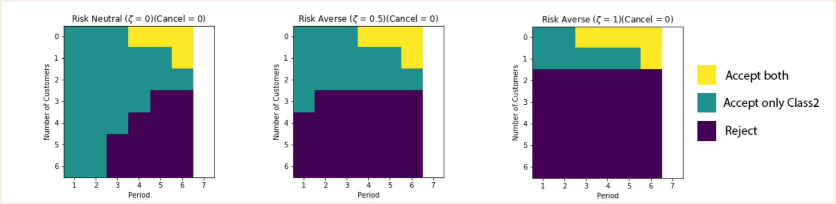
Parameter Setup

```
C = 2  # capacity of the plane
K = 7  # number of periods
kappa = 3  # fixed return of cancellation
beta_s = 0.95  # show-up probability
m = 2  # number of classes
r = np.array([5, 10])  # price of a fare class i ticket
theta = 12  # denied boarding (overbooking cost)

def set_values_accordingly(K):
    global c, p0, p
    c = np.linspace(0.01, 0.05, K+2)
    p0 = np.linspace(0.2, 0.1, K+2)
    p = np.array([p0.tolist(), ((1 - p0) / 2).tolist(), ((1 - p0)
         / 2).tolist()])  # we have two classes with equal
        # request probablitities
```

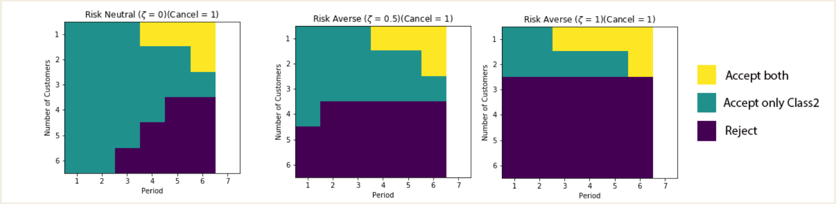
Computational Experiments

0 Cancellation



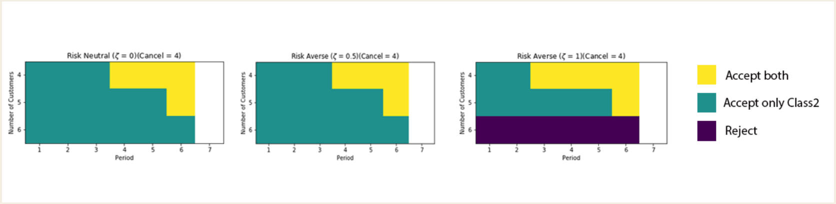
Computational Experiments

1 Cancellation



Computational Experiments

4 Cancellation



Computational Experiments

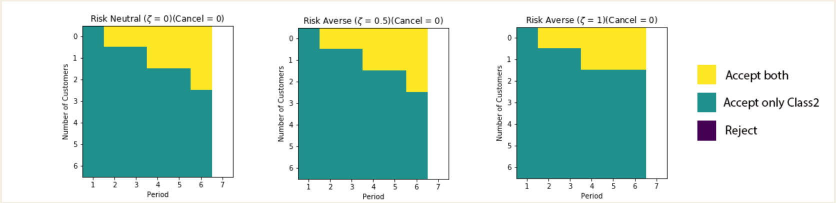
A Different Parameter Setup

```
C = 2  # capacity of the plane
K = 7  # number of periods
kappa = 4.5  # fixed return of cancellation
beta_s = 0.7  # show-up probability
m = 2  # number of classes
r = np.array([5, 10])  # price of a fare class i ticket
theta = 12  # denied boarding (overbooking cost)

def set_values_accordingly(K):
    global c, p0, p
    c = np.linspace(0.01, 0.05, K+2)
    p0 = np.linspace(0.2, 0.1, K+2)
    p = np.array([p0.tolist(), ((1 - p0) / 2).tolist(), ((1 - p0)
         / 2).tolist()])  # we have two classes with equal
        # request probablitities
```

Computational Experiments

0 Cancellation



Computational Experiments

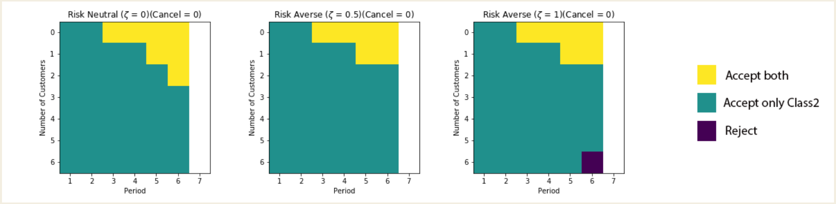
A Different Parameter Setup

```
C = 2  # capacity of the plane
K = 7  # number of periods
kappa = 4.5  # fixed return of cancellation
beta_s = 0.7  # show-up probability
m = 2  # number of classes
r = np.array([5, 10])  # price of a fare class i ticket
theta = 14  # denied boarding (overbooking cost)

def set_values_accordingly(K):
    global c, p0, p
    c = np.linspace(0.1, 0.15, K+2)
    p0 = np.linspace(0.2, 0.1, K+2)
    p = np.array([p0.tolist(), ((1 - p0) / 2).tolist(), ((1 - p0)
         / 2).tolist()])  # we have two classes with equal
        # request probablitities
```

Computational Experiments

0 Cancellation



References I

[1] M. Shaked and J. G. Shanthikumar, Stochastic Orders and Their Applications. Academic Press, San Diego, California, 1994.

[2] Lippman, Steven A., and Shaler Stidham Jr. "Individual versus social optimization in exponential congestion systems." Operations Research 25.2 (1977): 233-247.

[3] Subramanian, Janakiram, Shaler Stidham, and Conrad J. Lautenbacher. "Airline Yield Management with Overbooking, Cancellations, and No-Shows." Transportation Science 33.2 (1999): 147-67.

[4] Aydın, Nurşen, S. Ilker Birbil, J. B. G. Frenk, and Nilay Noyan. "Single-Leg Airline Revenue Management with Overbooking." Transportation Science 47.4 (2013): 560-83.

References II

[5] Artzner, Philippe, F. Delbaen, J.M. Eber and D. Heath. "Coherent Measures of Risk.", Mathematical Finance 9.4 (1999): 203-228.

Thank you for listening.

Appendix

Function of Model

```
def mo_ra_dyn(x, y):
        @memoize
        def V(k, n):
            if k == K:
                mu = -kappa*n*c[k] - theta*binom(n, beta_s*(1-c[k
                    ])).expect(func=lambda x: [max(0, i-C) for i
                    in x])
                ret = mu - zeta*binom(n, c[k]).expect(func=lambda
                     x: [binom(n-z, beta_s).expect(func=lambda y:
                     [max(0, mu +kappa*z+theta*max(0, t-C)) for t
                     in y]) for z in x])
```

Appendix

Function of Model

```
 elif k<K:
               mu = -kappa*n*c[k] + p[0, k]*binom(n, 1-c[k]).
                   expect(func=lambda x: [V(k, j) for j in x]) +
                     sum(p[i, k]*binom(n, 1-c[k]).expect(func=
                   lambda x: [max(r_[i]+V(k, j+1), V(k, j)) for
                   j in x]) for i in range(1, m+1))
               ret = mu -  zeta*binom(n, 1-c[k]).expect(func=
                   lambda x: [sum(p[i, k]*max(0,  mu+kappa*(n-t)
                   -max(r_[i]+V(k, t+1), V(k, t))) for i in
                   range(0, m+1)) for t in x] )
           return ret

       return V(x, y)
```

Appendix

Retrieving Decision Matrix

```
for n in range(K):
    for t in range(1, K):
        for i in range(n+1):
            decision_matrix [n][t-1][i] = decide([r_[1]+
                mo_ra_dyn(t+1, n+1-i),  mo_ra_dyn(t+1, n-i)])
```

Appendix

Calculation of Probability and Rewards in Last Node

```
def f(t, n, pro, rev):
    if t == K:
        for i in range(n+1):
            for s in range(n-i+1):
                rev_ = rev - kappa*i - theta*max(0, s-C)
                pro_ = pro*binom(n, c[t+1]).pmf(i)*binom(n-i,
                     beta_s).pmf(s)
                table.append([pro_, rev_])
```

Appendix

Calculation of Probability and Rewards in Last Node

```
if t < K:
            #p0
            for i in range(n+1):
                f(t+1, n-i, pro*p[0, t+1]*binom(n, c[t+1]).pmf(i)
                     , rev-kappa*i)
            #p1
            for i in range(n+1):
                if decision_matrix[n][t-1][i] == "accept":
                    f(t+1, n-i+1, pro*p[1, t+1]*binom(n, c[t+1]).
                        pmf(i), rev+r_[1]-kappa*i)
                elif decision_matrix[n][t-1][i] == "reject":
                    f(t+1, n-i, pro*p[1, t+1]*binom(n, c[t+1]).
                        pmf(i), rev-kappa*i)
```

Appendix

Calculation of Probability and Rewards in Last Node

```
#p2
for i in range(n+1):
    if decision_matrix_2[n][t-1][i] == "accept":
        f(t+1, n-i+1, pro*p[2, t+1]*binom(n, c[t+1]).
            pmf(i), rev+r_[2]-kappa*i)
    elif decision_matrix_2[n][t-1][i] == "reject":
        f(t+1, n-i, pro*p[2, t+1]*binom(n, c[t+1]).
            pmf(i), rev-kappa*i)
```