Constrained & unconstrained opt.

↳ obj fn

↳ 1st order deriv info, sometimes 2nd.

↳ Hessian info, typically costly to compute.

C^1 → 1st order deriv is a cont function.

General form

Model: $g(w|x)$ — w: set of params to determine; x: input vars

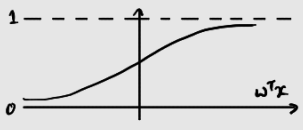
Agenda: $\min_{w \in \mathbb{R}^n} f(w) = \frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} - g(w|x^{(i)})\right]^2$

actual − predictions made by the model

1st order necessary condition: Let w^* be a minimiser of $f(w) \in C^1$, then $\nabla f(w^*) = 0$.

Unconstrained opt w/o exact soln
e.g. Logistic model

$$g(w|x) = \frac{e^{w^T x}}{1+e^{w^T x}}$$



View training data points as independent draws from the same population.

∴ choose w to maximise joint probability: MLE

$$\max_w h(w) = \prod_{i=1}^{m}\left(\frac{e^{w^T x^{(i)}}}{1+e^{w^T x^{(i)}}}\right)^{y^{(i)}}\left(\frac{1}{1+e^{w^T x^{(i)}}}\right)^{1-y^{(i)}}$$

BUT when $m \to \infty$, $h(w) \to 0$ ∴ opt problem hard to solve.

SOLUTION: Monotone transformation of the obj function. New & OG prob same.

$$\max_w h(w) = \prod_{i=1}^{m} p_i^{y^{(i)}}(1-p_i)^{1-y^{(i)}} \iff \max_w \ln\left(\prod_{i=1}^{m} p_i^{y^{(i)}}(1-p_i)^{1-y^{(i)}}\right)$$

Obj fn: $\min_w f(w) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\ln p_i + (1-y^{(i)})\ln(1-p_i)\right]$, where $p_i = \frac{e^{w^T x^{(i)}}}{1+e^{w^T x^{(i)}}}$

Gradient descent: Steepest descent in Log Reg

Step 1: Initialise random vector $w(0)$.

Step 2: For iter $k = 0, 1, 2, \ldots$

compute gradient:

$$\nabla f(w(k)) = -\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\left(\frac{1}{1+e^{w^T x^{(i)}}} + y^{(i)} - 1\right)$$

update:

$$w(k+1) = w(k) - \alpha \nabla f(w(k))$$

↳ Fixed learning rate can be found via LINE SEARCH.

Termination: stop when $\|\nabla f(w(k))\| \leq \epsilon$ OR k is beyond a cutoff.

- The −ve gradient of a function is its steepest desc dir.

$$\underbrace{f(w(k)+\eta\hat{v})}_{w(k+1)} - f(w(k)) \approx \underbrace{\eta \nabla f^T(w(k))\hat{v}}_{\text{directional deriv @ } w(k) \text{ along the dir } \hat{v}} \geq -\eta\|\nabla f(w(k))\|$$

(1st order Taylor approx)

$w(1) - w(0) = \Delta w = \eta\hat{v}$ → dir vector

$$\Delta w = -\eta\frac{\nabla f(w(0))}{\|\nabla f(w(0))\|}$$

$$\Delta w = -\alpha \nabla f(w(0))$$

η should ↑ w the magnitude of the slope.

comments
- may stop prematurely $\|\nabla f(w(t))\| < \epsilon$
- set target min value for $f(w)$ → but may end up running algo forever.
- set limit on total no. of steps in algo.

Steepest desc w Exact line search

Exact Line search

Step 2: For iter $k = 0, 1, 2, \ldots$

Pick a descent dir $p(k)$:

E.g. $p(k) = -\nabla f(w(k))$

Pick step size α by solving:

$$\alpha_k = \arg\min_{\alpha > 0} \phi(\alpha) = f(w(k) + \alpha p(k))$$

update:

$$w(k+1) = w(k) + \alpha_k p(k)$$

Induce Global convergence. Starting from any init point, this algo will bring you to a stat. point (provided the grad is Lipschitz continuous).

In practice, seldom solve line search in exact form. Try to ensure enough ↓ in obj fn value in line search step w/o spending time to solve it to optimality.

Steepest Descent w/ Inexact Line Search

Inexact line search
- Formulate criterion that assures that the step size is neither too large nor too small.
- pick good init step size $\alpha^{(0)}$.
- construct a sequence of updates of the step size $\alpha^{(k)}$ that satisfies the above criterion aft v. few iters.

Inexact line search: Backtracking

step 1: Initialise $\alpha^{(0)} = 1$ (or some +ve no.) and set $t = 0$.

step 2: until $f(w(k) + \alpha^{(t)} p(k)) < f(w(k))$ do:
- i) set $\alpha^{(t+1)} = \rho \alpha^{(t)}$, where $\rho \in (0,1)$.
- ii) $t \mathrel{+}= 1$.

Does not prevent α from being too large!

step 3: set $\alpha_k = \alpha^{(t)}$

Backtracking w/ Wolfe Condition

Armijo + curvature

step 2: until:
- $f(w(k) + \alpha^{(t)} p(k)) < f(w(k)) + \alpha^{(t)} c_1 \cdot p^T(k) \nabla f(w(k)) \triangleq l(\alpha)$
- $-\nabla f(w(k) + \alpha^{(t)} p(k))^T p(k) \geq c_2 \cdot p^T(k) \nabla f(w(k))$

where $0 < c_1 < c_2 < 1$, do:
- i) set $\alpha^{(t+1)} = \rho \alpha^{(t)}$, where $\rho \in (0,1)$
- ii) $t \mathrel{+}= 1$

Comments
- no momentum — may converge to local min and fail to "look for" global min.

Steepest Descent w/ Momentum

update step: $w(k+1) = w(k) + v(k)$

$= w(k) - \alpha \nabla f(w(k)) + \gamma v(k-1)$

$\gamma = 0.8, 0.9$ → aka "friction".

$v(0) = 0$

- with momentum, prev gradients are averaged w/ exponentially decaying weights.

$$
\begin{aligned}
v(k) &= -\alpha \nabla f(w(k)) + \gamma v(k-1) \\
&= -\alpha \nabla f(w(k)) + \gamma [-\alpha \nabla f(w(k-1)) + \gamma v(k-2)] \\
&\vdots \\
&= \underbrace{-\alpha \nabla f(w(k))}_{\text{curr gradient}} \underbrace{- \gamma \alpha \nabla f(w(k-1)) - \gamma^2 \alpha \nabla f(w(k-2)) + \gamma^2 v(k-3)}_{\text{prev gradients}} \\
&\vdots \\
&= -\alpha \sum_{i=0}^{k} \gamma^{k-i} \nabla f(w(i)) + \gamma^k v(0) \\
&= -\alpha \sum_{i=0}^{k} \gamma^{k-i} \nabla f(w(i)) \quad \because v(0) = 0
\end{aligned}
$$

Nesterov's Accelerated Grad Desc (NAG): $w(k+1) = w(k) - \alpha \nabla f(w(k) + \gamma v(k-1)) + \gamma v(k-1)$

Another implementation of NAG: $w(k+1) = w(k) + \gamma [v(k) - v(k-1)] + v(k)$ where $v(k) = \alpha \nabla f(w(k)) + \gamma v(k-1)$

Stochastic Gradient Desc (same steps as steepest desc)

Computing full gradient is very costly. Need to compute m gradients. If m is large, ☹ $

$$f(w) = -\frac{1}{m}\sum_{i=1}^{m} h_i(w; x^{(i)}, y^{(i)})$$

$$\Rightarrow \nabla f(w) = -\frac{1}{m}[\nabla h_1(w) + \nabla h_2(w) + \dots + \nabla h_m(w)]$$

∴ Randomly pick a function $h_i(w)$ among $\{h_i(w)\}_{i=1}^m$ & compute $\nabla h_i(w)$.
OR pick random subset of functions (e.g. 10%) and compute their gradients & take average.

Intuition of picking random function $h_i(w)$

"Average direction": $\mathbb{E}[-\nabla h_i(w; x^{(i)}, y^{(i)})] \approx -\frac{1}{m}\sum_{i=1}^{m} \nabla h_i(w; x^{(i)}, y^{(i)}) = \nabla f(w)$

- In ML, since we often assume that all training data samples are indep random draws from the same joint dist (x,y), randomly picking the grad of 1 comp function ≡ grad of full function ∓ some random noise.

Termination condition: $\left\| \frac{1}{R}\sum_{i=1}^{R} \nabla h_i(w; x^{(i)}, y^{(i)}) \right\| \leq \varepsilon$, where $R < m$, as $R \geq 1000$ usually.
↳ num of component functions selected.

$\mathbb{E}\left[\frac{1}{R}\sum_{i=1}^{R} \nabla h_i(w; x^{(i)}, y^{(i)})\right] \approx -\frac{1}{m}\sum_{i=1}^{m} \nabla h_i(w; x^{(i)}, y^{(i)}) = -\nabla f(w)$, if R is large.

$\|w(k+1) - w(k)\| \leq \varepsilon$

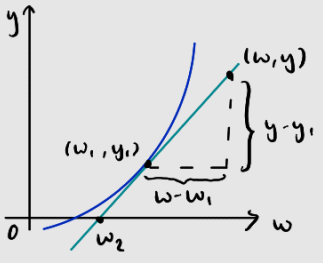
$\|f(w(k+1)) - f(w(k))\| \leq \varepsilon$

comments
- Takes more iter to converge than steepest desc, but since cheaper to compute the gradient in each iter, ∴ much faster overall.
- Randomization.
- Can apply fixed learning rate, e.g. $\alpha = 0.1$

Algos ∓ info beyond 1st gradient

- 1st order info is helpful in deriving descent direction: $p^T(k) \cdot \nabla f(w(k)) < 0$.
- 2nd order info (curvature info) tells you how big of a step you should take.
- Algos that use 2nd order info converge faster than only 1st info.

Newton's Method (same steps as steepest desc)



$f'(w) = \frac{y - y_1}{w - w_1} \iff y = f'(w)(w - w_1) + y_1$

when $y = 0$,

$w_2 = w_1 - \frac{y_1}{f'(w_1)} = w_1 - \frac{f(w_1)}{f'(w_1)}$

$\therefore w_{k+1} = w_k - \frac{f'(w_k)}{f''(w_k)}$ for solving $f'(w) = 0$.

Newton's method Optimization

Function: $f(w) \approx f(w(k)) + \nabla f_k^T (w - w(k)) + \frac{1}{2}(w - w(k))^T \nabla^2 f_k (w - w(k)) \triangleq q(w)$

For $q(w)$: $w^* = w(k) - [\nabla^2 f(w(k))^{-1}] \nabla f(w(k))$

w^*: global min

$[\nabla^2 f(w(k))^{-1}] \nabla f(w(k))$: this is direction p_k. ∴ step size $\alpha = 1$.

$\nabla^2 f_k$: Hessian matrix

$$\begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 f}{\partial w_1 \partial w_n} \\ \frac{\partial^2 f}{\partial w_2 \partial w_1} & \frac{\partial^2 f}{\partial w_2^2} & \cdots & \frac{\partial^2 f}{\partial w_2 \partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial w_n \partial w_1} & \frac{\partial^2 f}{\partial w_n \partial w_2} & \cdots & \frac{\partial^2 f}{\partial w_n^2} \end{bmatrix}$$

comments
- Fast convergence when you get to the min point.
- The obj fn may not be twice differentiable
- computation & storage for Hessian can be costly.
- Newton's direction is not always descent.

Newton's method vs Steepest descent

$w(k+1) = w(k) - [\nabla^2 f(w(k))^{-1}] \nabla f(w(k))$

knows everything about the curvature

$w(k+1) = w(k) - \alpha \cdot I \cdot \nabla f(w(k))$

completely agnostic abt the curve.

Quasi-Newton's method

$w(k+1) = w(k) - \alpha \cdot H(w(k)) \cdot \nabla f(w(k))$

- better convergence than I
- less expensive than inverse Hessian.

Steepest Desc After Transformation

Smooth convex function: $\min_w f(w)$

Rescale space: $w = Au$, A is positive definite & invertible.

$u = A^{-1}w$

New definitions: $g(u) = f(w)$, $g(u) = f(Au)$

$u^* = \arg\min_u g(u)$

$w^* = \arg\min_w f(w)$

$w^* = Au^* = 0$

- Affine transformation could greatly speed up steepest desc by making the problem less ill-conditioned.
- Rescale contour curves to smth that is "close" to perfect circles but balance the computation cost.

What is a good transformation

- Lemma: If matrix M is positive definite, then $M^{-1/2}$ exists and is also symmetric & strictly +ve eigenvalues.

All Quasi Newton methods: $w(k+1) = w(k) - \alpha_k \cdot B_k^{-1} \cdot \nabla f(w(k))$

→ If $B = \nabla^2 f(w)$, then it's Newton

- update B_k iteratively (allows rich B_k without recomputing from scratch every iter)
- Required: $B_k^{-1} \cdot \nabla f(w(k))$ easier to compute than Newton.
- Construct B_{k+1} s.t. $B_{k+1}[w(k+1) - w(k)] = \nabla f(w(k+1)) - \nabla f(w(k))$.

BFGS $O(n^2)$

In BFGS: $H_k = B_k^{-1}$

1. H_{k+1} is close to H_k
2. H_k is symmetric
3. H_{k+1} satisfies the secant equation $s_k = H_{k+1} y_k$.

Step 1: Initialise $w(0)$ and $H_0 = I$.

Step 2: For iter $k = 0, 1, 2, \ldots$

Repeat until some termination condition:

i) Pick descent direction to go $p(k) = -H_k \nabla f(w(k))$

$w(k+1) = w(k) - \alpha_k \cdot H_k \nabla f(w(k))$

BFGS direction is a desc dir if Wolfe condition holds.

ii) Pick step size using Wolfe condition.

iii) Update:

$s_k \triangleq w(k+1) - w(k)$ → diff in 2 consecutive values

$y_k \triangleq \nabla f(w(k+1)) - \nabla f(w(k))$ → diff in 2 consecutive gradients.

If $y_k^T s_k \geq \varepsilon_{skip} \|y_k\| \|s_k\|$, where ε_{skip} is some small number, e.g. 10^{-8}

$H_{k+1} = (I - \rho_k s_k y_k)^T H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$, where $\rho_k = \frac{1}{y_k^T s_k}$

Markov chain

Simulate a long process & compute the LR average dist.

- Limiting distribution — prob dist to which the Markov chain converges as the no. of steps $\to \infty$. ⟶ ⟵ stationary distribution (solve directly (eigenvalue problem))

$\lim_{t \to \infty} M^t p = \pi$

no. of iters — stochastic pagerank transition matrix.

$\pi = M \cdot \pi$

If you start the process w the stat dist π, then the probabilities of being in each state remains unchanged after any further transition steps.

- Long Run Average Dist (LR proportion of time)
  - refers to the fraction of time, as $t \to \infty$, that the Markov chain spends in each state i.

$\lim_{t \to \infty} \frac{1}{t} \sum_{k=0}^{t} P\{X_k = i\}$, where X_k is the state of the M.C. @ step k.

For finite state Markov chain, stat. dist ↔ LR Average dist.

Generating random variables

* with a known RV & a deterministic algorithm(), you can produce a new RV. random var X = algorithm(U) ← known dist.

Inverting Standard Normal CDF

Invert cdf to generate ANY distribution.
we want to write x as a function of u.

E.g. $u = F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt$

$x = F^{-1}(u)$

$X = F^{-1}(U)$

BUT:
Numerical integration → slow
Numerically find root → slow

Acceptance Region

Let X be the RV of interest, the rej method applies when we can express

$$F(x) \triangleq \mathbb{P}\{X \le x\} = \mathbb{P}\{V \le x \mid A\}$$

another RV; A: some "acceptance" event related to V.

- works well when not too many rejections occur.
- Better than inverse cdf if cdf is hard to invert.

To use some basic RV generator, 2 conditions:
1) Output of the transformation must coincide w the support of the target RV, X.
   ↳ set of points that the RV takes w non-0 prob.
2) Output of the function must have same statistical properties as the target RV X.
   ↳ uniquely determined by its cdf.

can handle $\mathcal{N}$ dist.

Just needs: - 2 uniform (non-standard) RNGs
- Pdf of the target random var.

- If you want to make 2 RVs "look similar" → squeeze / stretch the cdf of 1 RV, changing the density of the support.

Markov chain Monte Carlo (MCMC)

Monte Carlo method: Generating realizations of RVs to compute quantities.

Constructing such a Markov Chain: (i.e. how to construct the 1-step TM, M, s.t. it can induce the desired target dist π as the markov chain's stat dist)

Assuming finite support of our target dist e.g. $i = 1, 2, \ldots, K$

$$\pi = M \cdot \pi \leftrightarrow \sum_{i=1}^{K} M_{ji}\pi_i = \pi_j, \quad \forall j = 1, 2, \ldots, K \quad \rightarrow \text{HARD.}$$

M_{ji}: TM: prob of transitioning from i to j.
π_i: prob of being @ i one step before.
π_j: prob of being @ j currently.

↓

Detailed balance eqn: $M_{ji}\pi_i = M_{ij}\pi_j, \ \forall i,j$ → Goal: construct M s.t. it satisfies detailed balance equation. STILL HARD.

If you just provide some arbitrary 1-step TM, Q, then:
we can introduce α_{ij}: $\underbrace{\alpha_{ji}Q_{ji}}_{M_{ji}}\pi_i = \alpha_{ij}Q_{ij}\pi_j, \ \forall i,j$

$\alpha_{ij} = Q_{ji}\pi_i$
$\alpha_{ji} = Q_{ij}\pi_j$
↳ This is a probability.

$(Q_{ji}\pi_i \neq Q_{ij}\pi_j)$

Given target distribution π

Choose an arbitrary one-step transition matrix Q

t=0, pick an arbitrary state x_0

For t = 0,1,2,….

Using Q and x_t to simulate x^*

Generate one realization of standard uniform random variable $u \sim U(0,1)$

If $u < \alpha_{x_t, x^*} = Q_{x^*, x_t} \pi_{x^*}$

$x_{t+1} = x^*$

else

$x_{t+1} = x_t$

Markovian decision process

Defined by:

$S_t = s \in S$: state @ time t

a : Action in set $A(s)$

$R(s, a)$: reward of being in s & action a.

$p(s' | s, a) = \mathbb{P}\{S_{t+1} = s' \mid S_t = s, a\}$: prob of transitioning to s if we are in S_t and take action a.

Goal: Solve for optimal policy s.t.:

$$\max_{\pi} U^{\pi}(s) \triangleq \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \rho^t \cdot R(S_t, a_t) \mid S_0 = s\right] \leq \sum_{t=0}^{\infty} \rho^t R_{max} = \frac{R_{max}}{1-\rho}$$

- $\max_{\pi} U^{\pi}(s)$: LR expected total discounted reward from being @ s onwards under the opt policy
- ρ: discfactor $\in (0,1)$; $\rho = 1$ in finite horizon prob.
- $R(S_t, a_t)$: ST reward from being @ S_t & taking action a_t.

$$\pi^*_s \triangleq \arg\max_{\pi} U^{\pi}(s)$$

Policy : Mapping from state space to Action space
i.e. $\pi(s) = a$ for $s \in S$.

- optimal policy (π) is indep of the initial starting state for the discounted total expected reward, so we can write: $\pi^* \equiv \pi^*_s$
- Expected utility ($U(s)$) under the optimal policy still depends on the initial state: $U(s) \triangleq U^{\pi^*}(s)$.

Bellman Equation

$$U(s) = \max_{a \in A(s)} \left\{ R(s,a) + \rho \sum_{s'} p(s'|s,a)\, U(s') \right\}$$

with n possible states → there are n Bellman equations, one for each state.
The n equations contain n unknowns — the utilities of the states. ∴ solve to find $\{U(s)\}_{s \in S}$.

Value Iteration Algo

Step 1: Initialise $\{U_0(s)\}_{s \in S}$ with arbitrary values.

Step 2: set δ to be some large number.
while $\delta > \varepsilon$ (some small number):
For all states $s \in S$:

$$U_{n+1}(s) \leftarrow R(s) + \rho \max_{a \in A(s)} \sum_{s'} p(s'|s,a)\, U_n(s')$$

$$\delta \leftarrow \max_{s \in S} |U_{n+1}(s) - U_n(s)|$$

Contraction Mapping

A contraction is a function $F(x)$, s.t. given x & y,

$|F(x) - F(y)| \leq \gamma |x - y|$, where $\gamma < 1$ (output < input)

Theorem: If F is a contraction on a complete metric space, then F has a unique fixed point. (Through repeated applications of the contraction function)

- only 1 fixed point.
- when the fn is applied to any arg, the value must get closer to the fixed point.

MDP ∞ continuous time (∞ time horizon)

Goal: Find what is the optimal production policy?

capacity: μ / unit time Holding cost: h /product/unit time Demand: λ /unit time.
Lost sale cost: c /product

MDP formulation:
- no. of inventories in the stock $S \in \{0, 1, \dots, K\}$
- Action: $a \in A(s) = \begin{cases} \{\text{produce, idle}\}, & \text{if } s < K \\ \{\text{idle}\}, & \text{if } s = K \end{cases}$
- Decision Epochs: points in time at which decisions are made.
  - ↳ Demand arrival epoch
  - ↳ Product completion epoch.
- Transition Probabilities

$\mathbb{P}\{S' = (s-1)^+ \mid S = s, a = \text{idle}\} = 1$, for $0 \le s \le K$

$\mathbb{P}\{S' = (s-1)^+ \mid S = s, a = \text{prod}\} = \frac{\lambda}{\lambda+\mu}$

$\mathbb{P}\{S' = s+1 \mid S = s, a = \text{prod}\} = \frac{\mu}{\lambda+\mu}$

Production time $\sim \exp(\mu)$

$$\mathbb{P}\{X > Y\} = \int_0^\infty \mathbb{P}\{X > y \mid Y = y\} f_Y(y)\, dy$$
$$= \int_0^\infty \mathbb{P}\{X > y \mid Y = y\} \lambda e^{-\lambda y} dy$$
$$= \int_0^\infty e^{-\mu y} \lambda e^{-\lambda y} dy$$
$$= \frac{\lambda}{\lambda+\mu}$$

$\mathbb{P}\{X \le Y\} = \frac{\mu}{\lambda+\mu}$

Inter-arrival time $\sim \exp(\mu)$

- Utility function: $u_t(s)$
  - ↳ total expected cost under the opt pol when there are t more periods to go (decisions) + system state is s. (s products in warehouse)

Value Iteration Algo

Finding optimal pol that minimizes the LR average cost per period.

Step 1: Initialise $u_0(s) = 0$, $\forall s = 0, 1, \dots, K$.
set δ to be some large no.

Step 2: while $\delta > \varepsilon$:
For all states $s \in S$:

i) For $s = K$:

$$u_t(K) \leftarrow \left[h \cdot K \cdot \frac{1}{\lambda}\right] + u_{t-1}(K-1)$$

($\frac{1}{\lambda}$: Average inter-arrival time)

ii) For $0 < s < K$:

$$u_t(s) \leftarrow \min \begin{cases} \left[h \cdot s \cdot \frac{1}{\lambda}\right] + u_{t-1}(s-1), & \text{if } a = \text{idle} \\ \left[h \cdot s \cdot \frac{1}{\lambda+\mu}\right] + \frac{\lambda}{\lambda+\mu} \cdot u_{t-1}(s-1) + \frac{\mu}{\lambda+\mu} \cdot u_{t-1}(s+1), & \text{if } a = \text{prod} \end{cases}$$

($\frac{1}{\lambda+\mu}$: Average time when either event occurs; $\frac{\lambda}{\lambda+\mu}$: Prob of arrival prior to product completion.)

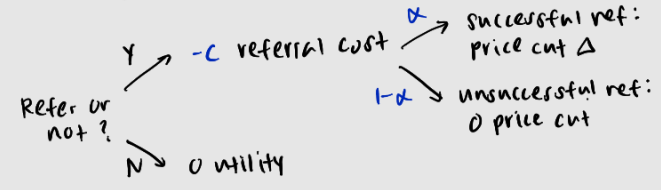
$$u_0(s) \leftarrow \min \begin{cases} \left[h \cdot 0 \cdot \frac{1}{\lambda}\right] + c + u_{t-1}(0), & \text{if } a = \text{idle} \\ \left[h \cdot 0 \cdot \frac{1}{\lambda+\mu}\right] + \frac{\lambda}{\lambda+\mu}\left[c + u_{t-1}(0)\right] + \frac{\mu}{\lambda+\mu} \cdot u_{t-1}(1), & \text{if } a = \text{prod} \end{cases}$$

$\delta \leftarrow \beta - \alpha$, where $\alpha = \min_{s \in S} u_t(s) - u_{t-1}(s)$

$\beta = \max_{s \in S} u_t(s) - u_{t-1}(s)$

- If the algo stops, then the LR average cost $= \frac{\beta + \alpha}{2}$ //

Help & Haggle



Consumer's Decision

@ start of game, firm announces product price P and the distribution of the random price cut (as proportion of the OG price), $\Delta \sim F(\cdot)$

LT reward: $U_t(S_t) = \max_{a \in A(S)} \{ R(S_t, a) + \int_{s \in S} \mathbb{P}\{s \mid S_t, a\} U_{t+1}(s) \}$

ST reward: $R(S_t, a) = -c \cdot \mathbb{1}_{\{a=1\}}$ → If $a=1$, then $-c$. If $a=0$, then 0

Transition probabilities:
- $\mathbb{P}\{S_{t+1} = p \mid S_t = p, a=0\} = 1$ — No ref
- $\mathbb{P}\{S_{t+1} = p \mid S_t = p, a=1\} = 1-\alpha$ — Yes ref, unsuccessful
- $\mathbb{P}\{S_{t+1} = p-x \mid S_t = p, a=1\} = \alpha \cdot \mathbb{P}\{\Delta = x\}, \forall x \in [0,1]$ — Yes ref, successful! (↳ price change)

Customer Decision: For $p > 0$ and $t \in \{0, 1, 2, ..., T-1\}$

$$U_t(p) = \max \left\{ U_{t+1}(p), \; -c + (1-\alpha) U_{t+1}(p) + \alpha \int_0^1 U_{t+1}(p-x) dF(x), \right\},$$

Never refer | Refers, unsuccessful | Refers, successful

with boundary condition: (T periods)

$$\begin{cases} U_t(p) = v, & \text{for } p \le 0, \text{ and } t \in \{0,1,2,...,T\}, \\ U_T(p) = (v-P)^+, & \text{for } p > 0. \end{cases}$$

→ consumer wins, product is free.
→ consumer loses, price still positive.
↳ price @ customer's valuation

Let $a_t^{(P,F)}(p)$ be the decision of the consumer at the start of period $t \in \{0, 1, ..., T-1\}$

$$a_t^{(P,F)}(p) = \begin{cases} 1, & \text{if } U_{t+1}(p) \le -c + (1-\alpha) U_{t+1}(p) + \alpha \int_0^1 U_{t+1}(p-x) dF(x), \\ 0, & \text{otherwise.} \end{cases}$$

↳ initial state = initial price

↳ Action a depends on init state (p) + time period (t)

Firm's Decision

At the start of the game, firm solves $\max_{P \ge 0, \; F(\cdot)} \pi_0^{(P,F)}(100\%)$, where

$$\pi_t^{(P,F)}(p) = a_t^{(P,F)}(p) \left[r + (1-\alpha) \pi_{t+1}^{(P,F)}(p) + \alpha \int_0^1 \pi_{t+1}^{(P,F)}(p-x) dF(x) \right] + [1 - a_t^{(P,F)}(p)] \pi_{t+1}^{(P,F)}(p),$$

where $a_t^{(P,F)}(p) = \begin{cases} 1, & \text{if } U_{t+1}(p) \le -c + (1-\alpha) U_{t+1}(p) + \alpha \int_0^1 U_{t+1}(p-x) dF(x), \\ 0, & \text{otherwise.} \end{cases}$

with boundary condition:

$$\begin{cases} \pi_t^{(P,F)}(p) = -k, & \text{for } p \le 0, \text{ and } t \in \{0,1,...,T\}, \\ \pi_T^{(P,F)}(p) = (P-k) \mathbb{1}_{\{v \ge P\}}, & \text{for } p > 0. \end{cases}$$

(Cost on the firm's side)

One can prove $P^* = v$.
set product price @ customer's valuation.

So we can focus on F^*.

friends also help.

$\alpha = 1$: Deterministic discount Δ (%)

small referral cost, can refer up to T times.

If $c < \frac{v}{T}$ ($\Leftrightarrow T = \min\{T, \frac{v}{c}\}$)

discount: $\Delta = \frac{1}{T}$

consumer utility: $v - cT > 0$

☹
- consumer surplus not fully extracted
- referrals are over-compensated

☺
- maximise social reach

Large ref cost, customer only refers $\frac{v}{c}$ times at most, although campaign lasts for T periods.

If $c \ge \frac{v}{T}$ ($\Leftrightarrow \frac{v}{c} = \min\{T, \frac{v}{c}\}$)

Firm's strat to set price cut at. $\Delta = \frac{1}{\lfloor v/c \rfloor}$

consumer utility: $v - c\lfloor v/c \rfloor \ge 0$

☹
- consumer surplus not fully extracted
- referrals are over compensated
- social reach $r\lfloor v/c \rfloor$ falls short of its potential $r(v/c)$

} For non-integer v/c

$\alpha = 1$: Randomised discount Δ (%) → rand from price cuts

$$\Delta = \begin{cases} \frac{1}{T} - \varepsilon & \text{small w.p. } (1-\frac{cT}{v})\frac{1}{T} \\ \frac{1}{T} + (T-1)\varepsilon & \text{big w.p. } 1-(1-\frac{cT}{v})\frac{1}{T} \end{cases}$$

where $\varepsilon \in \left(0, \frac{1}{T(T-1)^2}\right)$

consumer refers in all T periods but NOT guaranteed to win.
consumer expected utility = 0

- maintains max social reach rT while extracting all consumer surplus.
- Transfers value from the consumer to the firm.

$$\Delta = \begin{cases} \frac{1}{\lfloor n \rfloor + 1}, & \text{small w.p. } 1-(\lfloor n \rfloor + 1 - n)^{\frac{1}{\lfloor n \rfloor}} \\ \frac{1}{\lfloor n \rfloor}, & \text{big w.p. } (\lfloor n \rfloor + 1 - n)^{\frac{1}{\lfloor n \rfloor}} \end{cases}$$

where $n = v/c$

consumer only refer $\lfloor n \rfloor$ under deterministic scheme.
Consumer guaranteed to win but may refer for either $\lfloor n \rfloor$ or $\lfloor n \rfloor + 1$ periods.
consumer expected utility = 0

- Transfers value from the consumer to the firm.
- creates value by expanding social reach from $r\lfloor n \rfloor$ to rn.

$\alpha < 1$: Deterministic price cut

$U_t^N(S_t)$: value of being in state S_t at time t and following the optimal policy from time t onward, given the referral target being set by the firm is N.

Firm's Decision: Choose the referral target N to maximize $\pi_0^N(0)$ Expected payoff when the game starts

For $s \in \{0, 1, ..., N-1\}$ and $t \in \{0, 1, 2, ..., T-1\}$ Initial s = 0 (customer haven't refer yet)

$$\pi_t^N(s) = a_t^N(s)\left[r + (1-\alpha)\pi_t^N(s) + \alpha\pi_t^N(s+1)\right] + \left[1 - a_t^N(s)\right]\pi_t^N(s),$$

with boundary condition:

$$\begin{cases} \pi_t^N(N) = -k, & \text{if } t \in \{0, 1, ..., T\}, \\ \pi_T^N(s) = v - k, & \text{if } s \in \{0, 1, ..., N-1\}. \end{cases}$$

Customer Decision: For $s \in \{0, 1, ..., N-1\}$ and $t \in \{0, 1, 2, ..., T-1\}$

$$U_t^N(s) = \max\left\{U_{t+1}^N(s),\ -c + (1-\alpha)U_{t+1}^N(s) + \alpha U_{t+1}^N(s+1)\right\},$$

with boundary condition:

$$\begin{cases} U_t^N(N) = v, & \text{for } t \in \{0, 1, 2, ..., T\}, \\ U_T^N(s) = 0, & \text{for } s \in \{0, 1, 2, ..., N-1\}. \end{cases}$$

$$a_t^N(s) = \begin{cases} 1, & \text{if } U_{t+1}^N(s) \le (1-\alpha)U_{t+1}^N(s) + \alpha U_{t+1}^N(s+1) - c, \\ 0, & \text{otherwise.} \end{cases}$$