MARKOV CHAINS

Example

- Inventory in stock at beginning of day t, for t = 0,1,2,…,
  - X_t - these are the states
  - One possible sample path
    - X_0=3, X_1=0, X_2=4, X_3=1
  - What is the probability of this sample path?

$$P(X_3 = 1, X_2 = 4, X_1 = 0, X_0 = 3) = P(X_3 = 1 \mid X_2 = 4, X_1 = 0, X_0 = 3)$$
$$\times P(X_2 = 4 \mid X_1 = 0, X_0 = 3) \times P(X_1 = 0 \mid X_0 = 3) \times P(X_0 = 3)$$

Example

- The process gets pretty cumbersome!
- Under Markov property

$$
\begin{aligned}
P(X_3 = 1, X_2 = 4, X_1 = 0, X_0 = 3) = P(X_3 = 1 \mid X_2 = 4) \\
\times P(X_2 = 4 \mid X_1 = 0) \\
\times P(X_1 = 0 \mid X_0 = 3) \\
\times P(X_0 = 3)
\end{aligned}
$$

- The conditional probability of tomorrow's inventory level depends only on what we have today
  - Not on entire inventory history!

Required Problem Knowledge

- All possible states of a system
  - Status of the system at some point in time
- The probabilities of transitioning between states
  - Represent the probability of changing from a state to another state at some point in time.
    - Also called "one-step" transition probabilities
  - Stationary transition probabilities
    - Does not depend on time

$$p_{ij} = P\left(X_{t+1} = j \mid X_t = i\right) = P\left(X_1 = j \mid X_0 = i\right)$$

Required Problem Knowledge

- Initial condition
  - Unconditional probability of each state at the beginning of process
  - The probability that we start in state *i*

$$P\left(X_0 = i\right)$$

Properties of Markov Chains

- Stochastic process with the Markovian property

$$p_{ij} = P\left(X_{t+1} = j \mid X_t = i\right) = P\left(X_1 = j \mid X_0 = i\right)$$

transition probability

- History does not matter – except the last step

$$P\{X_{t+1} = j \mid X_0 = k_0, X_1 = k_1, \cdots, X_{t-1} = k_{t-1}, X_t = i\}$$
$$= P\{X_{t+1} = j \mid X_t = i\}$$

Example

- An employee either shows up for work or is absent.
  - If an employee shows up today and yesterday, with probability .9 she shows up tomorrow.
  - If an employee does not show up today nor yesterday, with probability .2 she shows tomorrow.
  - If an employee shows up today, but not yesterday, with probability .3 she shows up tomorrow.
  - If an employee showed up yesterday, but not today, with probability .8 she shows up tomorrow.

Example

- State = 0 if an employee shows up today and yesterday
- State = 1 if an employee shows up today but not yesterday
- State = 2 if an employee does not show up today, but did yesterday
- State = 3 if an employee does not show up either today or yesterday

Example

- Transition matrix *P*

$$\begin{pmatrix} .9 & 0 & .1 & 0 \\ .3 & 0 & .7 & 0 \\ 0 & .8 & 0 & .2 \\ 0 & .2 & 0 & .8 \end{pmatrix}$$

Main Result

- Suppose we start with an initial probabilistic vector *x*
- The probability that after *n* steps we are in state *j* is

$$\left(xP^n\right)_j$$

  - The *j*-th coordinate in

$$xP^n$$

Example

- If an employee shows up today but not yesterday (state 1)

$$x = (0,1,0,0)$$

- $xP = (.3\ \ 0\ \ .7\ \ 0)$ = distribution of what state the employee will be in on the following day
- $(xP)P = xP^2 = (.27\ \ .56\ \ .03\ \ .14)$ = distribution of what state the employee will be in two days later

Remarks

- $x = \underbrace{(0, 0, \cdots, 0, 1, 0, 0, \cdots 0)}_{i\text{th coordinate}}$
  - Start in state *i*
- Probability of being in state *j* after *n* steps

$$P_{ij}^{n}$$

- If *x* is a 'true probabilistic' vector
  - Start in a random state
  - Each initial state has its corresponding probability

Chapman-Kolmogorov Equations

- For any *k*

$$p_{ij}^{(n)} = \sum_{h=1}^{m} p_{ih}^{(k)} p_{hj}^{(n-k)}$$

- The probability of being in state *j* after *n* steps
  - Decomposed into the probability of being in state *h* after *k* steps
  - Then moving from state *h* to *j* in the remaining *n-k* steps
  - Provided we account for all possible intermediate states *h*

Steady-State Probabilities

- For some Markov chains
  - Long-run behavior is independent of how the process begins
- Stationary probability

$$\pi_j = \lim_{n \to \infty} p_{ij}^{(n)}$$

- Keep walking for a long time and count at what state stopped

Steady-State

- Equations

$$\underbrace{\pi = \pi P}_{a\ vector-matrix\ product} \qquad \underbrace{\sum_{all\ j} \pi_j = 1}_{a\ vector\ norm}$$

- If we have 1,...,*M* states, then we have *M*+1 equations in *M* unknowns.
- π is a left eigenvector, with eigenvalue 1.
  - Also a probability distribution!
- Can use typical matrix methods to solve for π.
  - Gaussian elimination, decomposition, find the inverse, etc.

Interpretation

- Station probability exists only for certain Markov chains
  - Ergodic Markov chains (aperiodic and irreducible)
- If the process runs for a long time, then π_i is the probability of being in state *i*
  - The process is stationary at this point
- This probability does not depend on the initial state
- The percentage of time spent in state *i*

Expected Costs

- Interpret π as the long-run % of time the process in state *i*
- Long-run costs or revenues associated with each state
  - Average cost

$$\textit{Expected cost / unit time} = \sum_{j=0}^{M} \pi_j C(j)$$

(sum over all costs, each weighted by its probability of occurrence)

Example

- Solving $\pi = \pi P$ leads to $\pi = (45, 21, 10) / 76$.

$$P = \begin{pmatrix} .7 & .1 & .2 \\ .5 & .5 & 0 \\ .3 & .6 & .1 \end{pmatrix}$$

- The linear equations to solve are

$$\pi_1 = .7\,\pi_1 + .5\,\pi_2 + .3\,\pi_3$$
$$\pi_2 = .1\,\pi_1 + .5\,\pi_2 + .6\,\pi_3$$
$$\pi_3 = .2\,\pi_1 + 0\,\pi_2 + .1\,\pi_3$$
$$1 = \pi_1 + \pi_2 + \pi_3$$
$$\pi_1 \geq 0,\ \pi_2 \geq 0,\ \pi_3 \geq 0$$

GRADIENT OPTIMIZATION

Loss Function

- $\sum_{x,y} l(x, y; w)$
  - x,y from train dataset
  - w model parameters
- Alternative viewpoint
  - Function that transforms x close to y
  - $f(x; w) \approx y$
- $\sum_{x,y} g(f(x; w), y) = \sum_{x,y} l(x, y; w)$
  - Function g captures penalty for not being perfect

$-\ln \frac{1}{1+e^{-wxy}}$

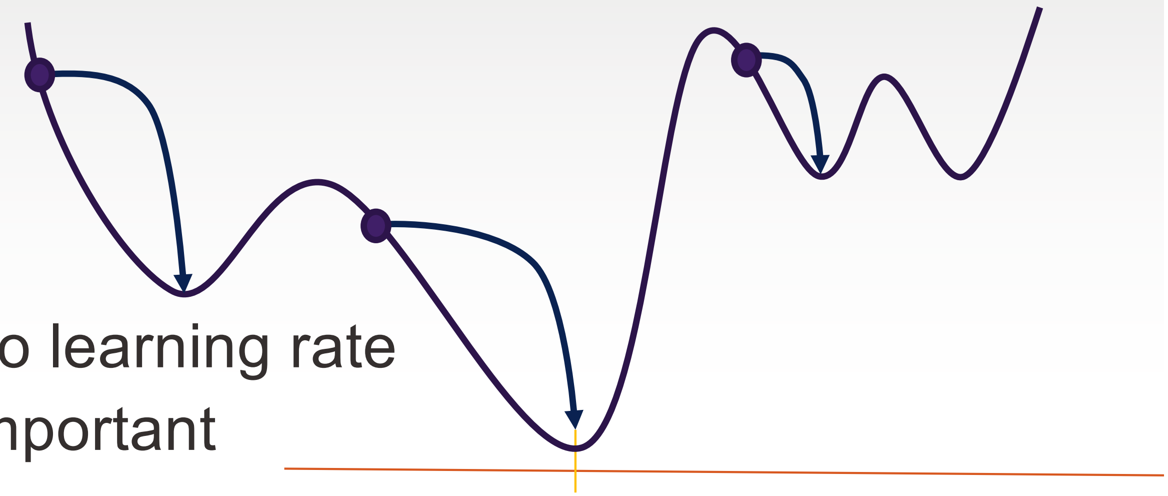
$(y - g(x; w))^2$

$\max(0, 1 - wxy)$

$$\min_{w} \sum_{xy} \ell(x, y; w)$$

Gradient Optimization

$$w_{m+1} = w_m - t_m \nabla f(w_m)$$



- Algorithm very sensitive to learning rate
- Starting point also very important
- Convergence
  - Certain learning rates lead to convergence
  - No guarantee to optimal solution
    - True only if function convex

Gradient

- Square error

$$\frac{\partial w}{\partial w_k} \sum_i (y_i - wx_i)^2 = -2 \sum_i x_{ik}(y_i - wx_i)$$

- Logistic regression

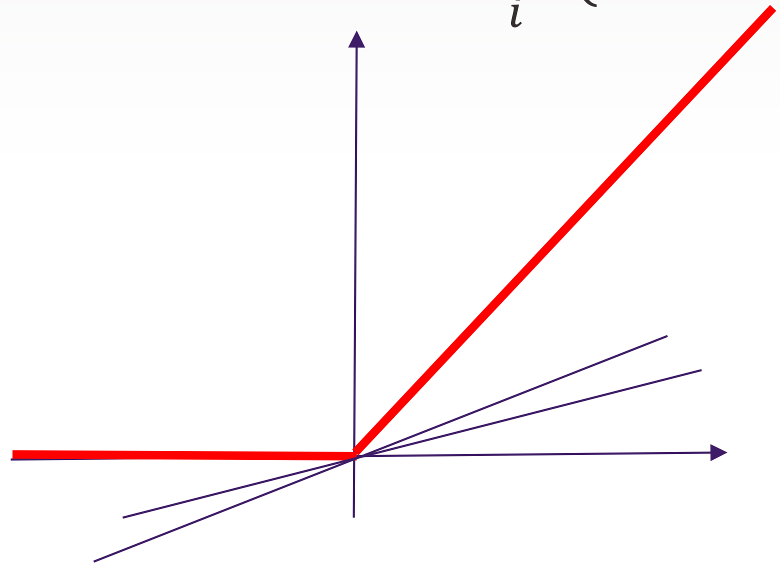
$$\frac{\partial w}{\partial w_k} \sum_i -ln(1 + \exp(-y_i w x_i))^{-1}$$

$$= \sum_i -y_i x_{ik} \exp(-y_i w x_i)\,(1 + \exp(-y_i w x_i))^{-1}$$

Gradient

- Hinge loss

$$\frac{\partial w}{\partial w_k} \sum_i \max(0, 1 - y_i w x_i) = \sum_i \begin{cases} -y_i x_{ik} & y_i w x_i < 1 \\ 0 & y_i w x_i \geq 1 \end{cases}$$



Gradient Optimization for ML

- Loss function involves all samples
  - Gradient is the sum of all gradients
  - Computationally intractable
- Solution
  - Randomly select subset S of samples
  - Compute the gradient for each selected sample
  - Average them
- Both solutions allow parallelization
  - Embarrassingly parallel

$$w_{m+1} = w_m - t_m \frac{1}{|s|} \sum_{i \in S} \nabla l(x_i, y_i; w_m)$$

Pros and Cons

Full gradient

High per iteration time

Low number of iterations

Most stable

Very low variance

Stochastic gradient descent

Lower per iteration time

Higher number of iterations

Harder to set learning rate

Higher variance

Adaptive: Adagrad

- Same learning rate for all weights
- Sparse feature (rare words in NLP) can be very important
  - Should get high learning rate
  - Likely to get small derivatives
- Weight j
  - Derivatives small (rare feature)
    - Larger learning rate to give them more importance
  - Derivatives large, small learning rate
- No longer moving in the gradient direction
- Learning rate always goes to zero
  - Not necessarily desirable

$$\lambda_t^j = \frac{\alpha}{\sqrt{\sum_{i=1}^{t-1} \nabla l(w_i)_j^2}}$$

Auxiliary Formula

$$u_t = \vartheta u_{t-1} + (1 - \vartheta) z_t$$

$$<=>$$

$$u_t = (1 - \vartheta)(z_t + \vartheta z_{t-1} + \vartheta^2 z_{t-2} + \cdots + \vartheta^t z_0)$$

- Exponential decay
- Can be easily tracked recursively

Adaptive: RMSprop

- Similar but weigh more recent iterates more
- Two parameters
  - Alpha as numerator
  - Theta for decay
- Updates can be made recursively

$$z_t^j = \nabla l(w_t)_j^2$$

$$lr_t^j = \frac{\alpha}{\sqrt{u_t^j}}$$