1 Bayes Network

Naive Bayes Effects are conditionally independent given a cause.
Bayesian network (G,P)**:** DAG with cond. prob. dist. $\mathrm{P}(X_s|Pa_{X_s})$. (G,P) defines joint distribution $\mathrm{P}(X_{1:n})=\prod_i \mathrm{P}(X_i|Pa_{X_i})$.
Specifying a BN Variables $X_1,\cdots,X_n$. Pick order. For all $i\in[n]$ find 1. min. subset $A\subseteq\{X_1,\ldots,X_{i-1}\}$ s.t. $X_i\perp X_{\bar{A}}|X_A$. 2. Specify/learn $\mathrm{P}(X_i|A)$.
BN defined this way are sound. Ordering matters a lot for compact. o. repr.!
Active Trails If for all consecutive triplets X,Y,Z.
- $X\to Y\to Z$ & Y is unobserved
- $X\leftarrow Y\leftarrow Z$ & Y is unobserved
- $X\leftarrow Y\to Z$ & Y is unobserved
- $X\to Y\leftarrow Z$ & Y or any of $Y's$ descendants is oberserved

X and Y *d-seperated by* Z iff no active trail exists with observations Z. Implies c.i.: d-sep$(X;Y|Z)\Rightarrow X\perp Y|Z$.

2 Inference

Typical Queries Cond. Distr., MPE ($\mathrm{argmax}_{e,b,a}\,\mathrm{P}(e,b,a|J=t,M=f)$), MAP ($\mathrm{argmax}_e \mathrm{P}(e|J=t,M=f)$)

2.1 Exact Inference

Variable Elimination
- Given BN and Query $\mathrm{P}(Q|E=e)$ (E = evidence variables)
- Choose an ordering of $X_1,\ldots,X_n$
- Set up initial factors: $fi=\mathrm{P}(X_i|P_{ai})$
- For $i=1:n,\ X_i\in X\setminus\{Q,E\}$

1. Collect and multiply all factors f that include X_i
2. Generate new factor by marginalizing out X_i
3. Add g to set of factors, $g=\sum_{x_i}\prod_j f_j$

- Renormalize $\mathrm{P}(q,e)$ to get $\mathrm{P}(q|e)=\frac{1}{\sum_q \mathrm{P}(q,e)}\mathrm{P}(q,e)$

Variable Elimination for Polytrees
Polytree: A DAG is a polytree iff dropping edge dir. $\to$ tree
- Pick root
- Orient edges towards the root
- Eliminate in topological ordering (descendants before parents)

Factor Graph of BN is bipartite graph with variables on one side and factors on the other.
Sum-Product / Belief Propagation Algorithm
- Initialize all messages as uniform distribution
- Until converged do

1. Pick some ordering on the factor graph edges (+directions)
2. Update messages according to this ordering

Messages from variable v to factor u:
$\mu^{(t+1)}_{v\to u}(x_v)=\prod_{u'\in N(v)\setminus\{u\}}\mu^{(t)}_{u'\to v}(x_v)$
Messages from factor u to variable v:
$\mu^{(t+1)}_{u\to v}(x_v)=\sum_{x_u\sim x_v} f_u(x_u)\prod_{v'\in N(u)\setminus\{v\}}\mu^{(t)}_{v'\to u}(x_{v'})$
3. Break once all messages change by at most ϵ
Intention: $\hat{\mathrm{P}}(X_v=x_v)\propto\prod_{u\in N(v)}\mu_{u\to v}(x_v)$
$\hat{\mathrm{P}}(X_u=x_u)\propto f_u(x_u)\prod_{v\in N(u)}\mu_{v\to u}(x_u)$
Belief Propagation on Trees (converges in two rounds)
- Factor graph of polytree is a tree!
- Choose one node as root
- Send messages from leaves to root, and from root to leaves

Variable Elimination for MPE
- Given BN and evidence $E=e$
- Choose an ordering of $X_1,\ldots,X_n$
- Set up initial factors: $f_i=\mathrm{P}(X_i|Pa_i)$
- For $i=1:n, X_i\notin E$

1. Collect and multiply all factors f_j that include X_i
2. Generate new factor by maximizing out $X_i, g_i=\max_{x_i}\prod_j f_j$
3. Add g to set of factors

- For $i=n:-1:1, X_i\notin E$: $\hat{x_i}=\mathrm{argmax}_{x_i} g_i(x_i,\hat{x}_{i+1:n})$

Example
$$\mathrm{argmax}_{e,b,a}\mathrm{P}(e,b,a|J,M)=\mathrm{argmax}_{e,b,a}\cancel{\tfrac{1}{Z}}\mathrm{P}(e,b,a,J,M)$$
$$=\mathrm{argmax}_{e,b,a}\mathrm{P}(e)\mathrm{P}(b)\mathrm{P}(a|e,b)\mathrm{P}(J|a)\mathrm{P}(M|a)$$
$$=\mathrm{argmax}_a\mathrm{P}(J|a)\mathrm{P}(M|a)\mathrm{argmax}_e\mathrm{P}(e)\mathrm{argmax}_b\mathrm{P}(b)\mathrm{P}(a|e,b)$$
$$a^*=\mathrm{argmax}_a P(J|a)P(M|a)g_e(a)$$
$$e^*=\mathrm{argmax}_e P(J|a^*)P(M|a^*)P(e)g_b(a^*,e)$$
Max-product Message Passing on Factor Graphs Messages from variable v to factor u:
$\mu^{(t+1)}_{v\to u}(x_v)=\prod_{u'\in N(v)\setminus\{u\}}\mu^{(t)}_{u'\to v}(x_v)$
Messages from factor u to variable v:
$\mu^{(t+1)}_{u\to v}(x_v)=\max_{x_u\sim x_v} f_u(x_u)\prod_{v'\in N(u)\setminus\{v\}}\mu^{(t)}_{v'\to u}(x_{v'})$
Retrieving MAP From Max-Product
- Define max-marginals: $P_{max}(X_v=x_v)=\max_{x\sim x_v}P(x)$
- For tree factor graphs, max-product computes max-marginals: $P_{max}(X_v=x_v)\propto\prod_{u\in N(v)}\mu_{u\to v}(x_v)$
- Can retrieve MAP sol. from these (must be careful with ties)

2.2 Approximate Inference

Problem: if BN contains loops. Loopy belief propagation will in general not converge $\to$ Sampling Based Inference
Monte Carlo Sampling from a BN (forward Sampling)
- Sort variables in topological ordering $X_1,\ldots,X_n$
- For $i=1$ to n do: Sample $xi\sim P(X_i|X_1=x_1,\ldots,X_{i-1}=x_{i-1})$

Computing Probabilities Through Sampling
Marginals: $P(w=t)\approx\frac{1}{N}\sum_{i=1}^N[w=t](x^{(i)}=\frac{Count(w=t)}{N}$
Conditionals: $P(C=t|W=t)=\frac{P(C=t,W=t)}{P(w=t)}\approx\frac{Count(W=t,C=t)}{Count(W=t)}$
Rejection Sampling "Normalen Würfel nehmen um Verteilung von 1..5 zu samplen, 6 wird jeweils ignoriert"
$\hat{P}(X_A=x_A|X_B=x_B)\approx\frac{Count(x_a,x_B)}{Count(x_B)}$
Throw away samples that disagree with x_B problematic if x_B is a rare event.
Markov Chain A MC is sequence of RVs, $X_1,\ldots,X_N$ with *Prior* $\mathrm{P}(X_1)$ & *transition probabilities* $\mathrm{P}(X_{t+1}|X_t)$ independent of t.
Ergodic: $\exists t\in\mathbb{N}$ s.t. every state is reachable from every state in exactly t steps. Then it has uniq., positive, stat. distribution.
$\exists$ uniq. stat. $\pi(x)>0$ *s.t.* $\forall x\lim_{N\to\infty}\mathrm{P}(X_N=x)=\pi(x)$ ind. of $\mathrm{P}(X_1)$.
Ergodic Thm: $\lim_{N\to\infty}\frac{1}{N}\sum_i f(x_i)=\sum_{x\in D}\pi(x)f(x)$, where D finite state space.

Sampling from MC Sample $x_1\sim\mathrm{P}(X_1), x_2\sim\mathrm{P}(X_2|X_1=x_1),\ldots,x_N\sim\mathrm{P}(X_N|X_{N-1}=x_{N-1})$. If simulated "sufficiently long", sample X_N is drawn "very close" to the stationary dist. π.

Algorithm 1: Gibbs Sampling: Random Order

```
Start with initial assignment x to all variables
Fix observed variables X_B to their observed values x_B
for t ← 1 to ∞ do
    Pick a variable i uniformly at random from {1,...,n}\B
    Set v_i = values of x, except for x_i
    Update x_i by sampling from P(X_i|v_i)
```

Satisfies detailed balance equation! For unnormalized Q $\forall x,x':\frac{1}{Z}Q(x)P(x'|x)=\frac{1}{Z}Q(x')P(x|x')$

Algorithm 2: Gibbs Sampling: Practical Variant

```
Start with initial ass. x^(0) to all (except the obs.) vars
Fix observed variables X_B to their observed values x_B
for t ← 1 to ∞ do
    x^(t) ← x^(t-1) (Load the previous sample)
    foreach variable X_i (except those in B) do
        Construct evidence:
          v_i = values of x^(t), except for
          x_i, since this is the variable that we want to sample.
        Sample x_i^(t) ~ P(X_i|v_i) (Update the value of the sample)
        x_i^(t) ← x_i^(t)
    Output the sample x^(t)
```

No detailed balance, but also has correct stationary distribution.
Ex. for Michi $\mathrm{P}(C|S,R,W=1)=\frac{\mathrm{P}(C,S,R,W=1)}{\sum_{c'}\mathrm{P}(C=c',S,R,W=1)}$.
Metropolis Hastings MCMC Algorithm
1. Proposal distribution $R(X'|X)$
- Given $X_t=x$, sample "proposal" $x'\sim R(X'|X=x)$
- Note: The performance of the algorithm will strongly depend on R.

2. Acceptance distribution:
- Suppose $X_t=x$
- With probability $\alpha=\min\left\{1,\frac{Q(x')R(x|x')}{Q(x)R(x'|x)}\right\}$ set $X_{t+1}=x'$
- With prob. $1-\alpha$, set $X_{t+1}=x$ (again to the current/same state).

2.3 Temporal models

Markov Chains
Markov assumption: $x_{t+1}\perp x_{1:t-1}|x_t\forall t$
Stationary asm.: $P(x_{t+1}=x|x_t=x')=P(x_{t'+1}|\ x_{t'}=x')\forall t,t'$
Hidden Markov Model/ Kalman Filters
$X_1,\ldots,X_T$: Unobserved (hidden) variables (called states)
$Y_1,\ldots,Y_T$: Observations
HMMs: X_i categorical, Y_i categorical (or arbitrary)
Kalman Filters: X_i,Y_i Gaussian distributions
Inference Tasks Filtering: $P(X_t|y_{1:t})$; Prediction: $P(X_{t+T}|y_{1:t})$; Smoothing: $P(X_t|y_{1:T})$ for $1\le t<T$; MPE: $\mathrm{argmax}_{X_{1:T}}P(X_{1:T}|\ y_{1:T})$

Bayesian Filtering Suppose we already have computed $P(X_t|y_{1,\dots,t})$ now want to efficiently compute $P(X_{t+1}|y_{1,\dots,t+1})$

- Start with $P(X_1)$
- At time t (Assume we have: $P(X_t|y_{1,\dots,t-1})$
- Conditioning: $P(X_t|y_{y1:t}) = \frac{1}{Z}P(X_t|y_{1:t-1})P(y_t|X_t)$
- $Z = \sum P(x,y_t|\ y_{1:t-1})$
- Prediction: $P(X_{t+1}|y_{y1:t}) = \sum_{x_t} P(X_t|y_{1:t})P(X_{t+1}|x_t)$

Computation is recursive (cost independent of t)!

Particle filtering Approximate the posterior at each time by samples (particles), which are propagated and reweighted over time. True distribution (possibly continuous): $P(x)$, N i.i.d. samples: $x_1,..x_N$; Represent: $P(x) \approx \frac{1}{N}\sum_{i=1}^{N}\delta_{x_i}(x)$. E.g. $\delta_{x_i}(x) := x_i = x$

- Suppose $P(X_t|\ y_{1:t}) \approx \frac{1}{N}\sum_{i=1}^{N}\delta_{x_{i,t}}$ (measurement)
- Prediction: Propag. each particle: $x_i' \sim P(X_{t+1}|x_{i,t})$ (movement)
- Conditioning: Weigh particles: $w_i = \frac{1}{Z}P(y_{t+1}|\ x_i')$
- Conditioning: Resample N particles: $x_{i,t+1} \sim \frac{1}{N}\sum_{i=1}^{N} w_i \delta_{x_i'}$

2.4 Probabilistic planning

How should we control the robot to maximize reward?

Markov Decision Process (MDP) – "controlled Markov chain": States $X=\{1,\dots,n\}$, actions $A=\{1,\dots,m\}$, transition probabilities $\mathrm{P}(x'|x,a) = \Pr[\text{next state } = x'|\text{Action } a \text{ in state } x)]$ and reward function $r(x,a)$.

Induced MC by Policy A deterministic policy π induces a *Markov Chain* $X_0,X_1,\dots,X_t,\dots$ with transition probabilities

$$P(X_{t+1}=x'|X_t=x) = P(x'|x,\pi(x))$$

Modes Finite horizon (T timesteps) or discounted rewards (∞ timesteps but discount factor γ).

Value of policy deterministic (fixed) policy $\pi : X \to A$.

$$\begin{aligned} V_\pi(x) &= J(\pi|X_0=x) = \mathbb{E}[\textstyle\sum_{t=0}^{\infty}\gamma^t r(X_t,\pi(X_t))|X_0=x] \\ &= \textstyle\sum_{x'} \mathrm{P}(x'|x,\pi(x))[r(x,\pi(x),x') + \gamma V_\pi(x')]. \\ &= r(x,\pi(x)) + \gamma \textstyle\sum_{x'} \mathrm{P}(x'|x,\pi(x))V_\pi(x') \end{aligned}$$

Given π, compute V_π exactly by solving linear system!

Greedy policy w.r.t. V

$$\begin{aligned} \pi_g(x) &= \operatorname{argmax}_a \textstyle\sum_{x'} \mathrm{P}(x'|x,a)(r(x,a,x') + \gamma V_\pi(x')) \\ &= \operatorname{argmax}_a r(x,a) + \gamma \textstyle\sum_{x'} \mathrm{P}(x'|x,\pi(x))V_\pi(x')). \end{aligned}$$

Bellmann Eq. Policy opt. $\iff$ greedy w.r.t. its ind. val. f. $V^*(x) = \max_a [r(x,a) + \gamma\sum_{x'} \mathrm{P}(x'|x,a)V^*(x')]$.

Recursion of Value Function

$$\mathbf{v}^\pi = \begin{pmatrix} V^\pi(1) \\ V^\pi(2) \\ \vdots \\ V^\pi(n) \end{pmatrix}, \quad \mathbf{T}^\pi = \begin{pmatrix} P(1|1,\pi(1)) & \cdots & P(n|1,\pi(1)) \\ P(1|2,\pi(2)) & \cdots & P(n|2,\pi(2)) \\ \vdots & \ddots & \vdots \\ P(1|n,\pi(n)) & \cdots & P(n|n,\pi(n)) \end{pmatrix}, \quad \mathbf{r}^\pi = \begin{pmatrix} r(1,\pi(1)) \\ r(2,\pi(2)) \\ \vdots \\ r(n,\pi(n)) \end{pmatrix}.$$

$$\mathbf{v}^\pi = \mathbf{r}^\pi + \gamma \mathbf{T}^\pi \mathbf{v}^\pi \iff \mathbf{v}^\pi = (\mathbf{I} - \gamma\mathbf{T}^\pi)^{-1}\mathbf{r}^\pi.$$

Algorithm 3: Policy Iteration

Start with an arbitrary (e.g., random) policy π
while *not converged* **do**
  (2) Compute value function $V^\pi(x)$ (solve LSE)
  (1) Compute greedy policy π_G w.r.t. V^π
  $\pi \leftarrow \pi_G$ (use the current greedy policy for the next iter.)

Converged when $\pi = \pi_G$. Guaranteed to monotonically improve, thus converging to an optimal policy in poly. steps. Iteration: $O(n^3)$-time.

Algorithm 4: Value Iteration (usualy learned off-policy)

foreach $x \in \mathcal{X}$ **do**
  $V_0(x) \leftarrow \max_a r(x,a)$ (Init)
Iteratively compute value function:
for $t \leftarrow 1$ **to** ∞ **do**
  foreach $x \in \mathcal{X}$ **do**
    foreach $a \in \mathcal{A}$ **do**
      $Q_t(x,a) = r(x,a) + \gamma\sum_{x'} P(x'|x,a)V_{t-1}(x')$
    $V_t(x) \leftarrow \max_a Q_t(x,a)$ (so called "Bellman Update")
  if $\|\mathbf{v}_t - \mathbf{v}_{t-1}\|_\infty < \epsilon'$ **then**
    break
Then choose greedy policy π_G w.r.t. V_t

Conv. to ϵ-optimal policy in poly. steps. Iteration: $O(n \cdot m \cdot a)$-time.

3 Learning BN

Learn from i.i.d. data: 1.) Learning structure (conditional independencies) 2.) Learning parameters (CPDs)

3.1 Parameter learning

$P(X_1,\dots,X_N) = \prod_{i=1}^{n} P(X_i|Pa_i,\theta_{i|\ Pa_i})$

Given: BN structure G & Data set D of complete observations For each variable X_i estimate: $\hat{\theta}_{X_i|\ Pa_i} = \frac{Count(X_i,Pa_i)}{Count(Pa_i)}$ (MLE)

We can also use a Beta prior (A priori knowledge).

3.2 Structure learning

Score based structure learning: scoring function S(G;D). Quantifies, for each structure G the fit to the data D.

$G^* = \operatorname{argmax}_G S(G;D)$; MLE: $S(G;D) = \max_\theta \log P(D|\theta,G)$

$\log P(D|\theta_{G,\text{ MLE for G}},G) = N\sum_{i=1}^{n}\hat{I}(X_i;Pa_i) + c.$

Problem: Optimal sol. for MLE is always the fully connected graph.

Mutual Inf (MI) $I(X_i;X_j) = \sum_{X_i,X_j} \mathrm{P}(X_i,X_j)\log\frac{\mathrm{P}(x_i,x_j)}{\mathrm{P}(x_i)\mathrm{P}(x_j)} \geq 0$

Empir. MI $\hat{\mathrm{P}}(X_i,\ X_j) = \frac{\#(X_i,X_j)}{N}$, $\hat{I}(X_i;\ X_j) = \sum_{x_i,x_j} \hat{\mathrm{P}}(x_i,x_j)\log\frac{\hat{\mathrm{P}}(x_i,x_j)}{\hat{\mathrm{P}}(x_i)\hat{\mathrm{P}}(x_j)}$

Entropy $H(X_i) = -\sum_{x_i} \mathrm{P}(x_i)\log \mathrm{P}(x_i) = \mathbb{E}_{x_i}[-\log \mathrm{P}(x_i)]$.

Properties of MI $I(X_i;X_j) = 0$ iff X_i,X_j independent. Symmetric. $I(X_A;X_B) = H(X_A) - H(X_A|X_B)$. $B \subseteq C \implies I(X_A;X_B) \leq I(X_A;X_C)$.

Bayesian Information Criterion (BIC) (Regularizing) $S_{BIC}(G) = \sum_{i=1}^{n}\hat{I}(X_i;Pa_i) - \frac{\log N}{2N}|G|$ where $|G|$ is # of parameters of G, n is # of variables, N is # of training examples.

Chow-Liu algorithm Given samples of $X_1,\dots,X_n$. Find BN with exactly one parent per variable. Gives opt. tree w.r.t. BIC.

- For each pair X_i,X_j of variables compute $\hat{P}(x_i,x_j)$
- Compute Mutual Information $\hat{I}(X_i,X_j)$
- Take graph K_n, weight edge $(X_i,X_j) = \hat{I}(X_i,X_j)$
- Find maximum spanning tree of graph $\to$ undirected tree
- Pick any variable as root and orient the edges away using BFS

4 Reinforcement Learning

"Learn mapping from (seq. of) actions to rewards"

4.1 Passive Reinforcement

Execute a set of trials in the environment using (fixed) policy π Reduction to a supervised problem. But does not exploit that values of states are not independent! (Bellman)

4.2 Active Reinforcement Learning

Not interested in fixed policy; need to decide action in every state. Fundamental Dilemma: Exploration-Exploitation.

- Always pick a random action - will eventually estimate all prob and rewards. May do extremely poorly.
- Always pick the best action according to current knowledge - quickly get some rewards but can get stuck in suboptimal action.

4.2.1 Model-based RL

Learn the MDP - optimize policy based on MDP.

Learning the MDP

Estimate transitions: $\hat{P}(X_{t+1}|X_t,A_t) = \frac{Count(X_{t+1},X_t,A_t)}{Count(X_t,A_t)}$

Estimate rewards: $\hat{r}(X=x,A=a) = \frac{\sum_{t:x_t=x,a_t=a} r_t}{Count(X=x,A=a)}$

ϵ_t **greedy** With probability ϵ_t: Pick random action; With probability $(1-\epsilon_t)$: Pick best action

R_{max} **Algorithm** Init: $r(x,a) = R_{max}$. $P(x^*|x,a) = 1$ where x^* is a "fairy tale" state: $r(x^*,a) = r_{max}\forall a$ Choose optimal π according to r,P

Repeat: Execute Pi. $\forall(x,a) \in visited$. update $r(x,a)$. Estimate $P(x'|x,a)$. After "enough" rewards, recompute π according to r,P.

Depends heavily on state space $O(|x|^3)$

4.2.2 Model-free RL

Estimate the value function directly.

Q-Learning $Q^*(x,a) = r(x,a) + \gamma \cdot \max_{a'}(Q^*(x',a'))$;

Algorithm Init Q matrix to zero (or R if *optimistic*).

- Select/Observe init state x
- Repeat until end state (restart after epoch):
- Select a according to policy (i.e. ϵ-greedy, or $\pi(x) = \operatorname{argmax}_a Q(x,a)$).
- Observe new state x' and reward $r(x,a,x')$.
- $Q^{(t+1)}(x,\ a) \leftarrow (1-\alpha_t)Q^{(t)}(x,\ a) + \alpha_t(r(x,\ a,\ x') + \gamma\max_{a'}Q^{(t)}(x',a'))$
- $Q^{(t+1)}(x,a') \leftarrow Q^{(t)}(x,a')\forall a \neq a'$, remaining actions stay same.
- $Q^{(t+1)}(x'',a') \leftarrow Q^{(t)}(x'',a')\forall x'' \neq x$, remaining states stay same.
- $x \leftarrow x'$

Depends on action space $O(|a|)$ (matrix size), i.e. iteration time is in $O(|a|)$.