Ryan Yee - CS 760 Final Note Sheet

Information Theory

Entropy: $H(Y) = -\sum_{y\in Y} \mathbb{P}(y)\log_2(\mathbb{P}(y))$
$H(Y|X) = \sum_{x\in X} \mathbb{P}(X=x)H(Y|X=x)$
Information Gain: $H(Y) - H(Y|X)$
Gain Ratio: $\frac{H(Y)-H(Y|S)}{H(S)}$

Evaluation

Accuracy: $\frac{TP+TN}{TP+FP+TN+FN}$, Error: 1 - Accuracy
Precision: $\frac{TP}{TP+FP}$, Recall: $\frac{TP}{TP+FN}$, FPR: $\frac{FP}{TN+FP}$

Estimation

Maximum Likelihood:
$\hat{\theta}_{MLE} = \arg\max_\theta \mathcal{L}(\theta; X)$
$\mathcal{L}(\theta; X) = \prod_{i=1}^n \mathbb{P}_\theta(x_i)$
Conditioned on X:
$\hat{\theta}_{MLE} = \arg\max_\theta \mathcal{L}(\theta; Y, X)$
$\mathcal{L}(\theta; Y, X) = \prod_{i=1}^n \mathbb{P}_\theta(y_i|x_i)$
Maximum a posteriori Probability:
$\hat{\theta}_{MAP} = \arg\max_\theta \prod_{i=1}^n p(x^{(i)}|\theta)p(\theta)$

Logistic Regression

Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)} = \frac{\exp(z)}{1+\exp(z)} \in (0,1)$
Properties: $1-\sigma(z) = \sigma(-z)$
$\sigma'(z) = \sigma(z)(1-\sigma(z))$
Cross Entropy Loss: $-[y\log\hat{y} + (1-y)\log(1-\hat{y})]$

Linear Regression

Loss: $\ell(f_\theta) = \frac{1}{n}||\mathbf{X}\theta - y||_2^2$
Gradient: $\nabla_\theta = \frac{1}{n}(2\mathbf{X}^T\mathbf{X}\theta - 2\mathbf{X}^Ty)$
Solution: $\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^Ty$
Ridge Loss: $\ell(f_\theta) = \frac{1}{n}||\mathbf{X}\theta - y||_2^2 + \lambda||\theta||_2^2$
Ridge Sol: $\theta = (\mathbf{X}^T\mathbf{X} + \lambda nI)^{-1}\mathbf{X}^Ty$

Regularization Techniques

1. Data augmentation (based on domain)
e.g. crop, rotate, thesaurus/back-translate
2. Adding noise (equivalent to weight decay)
3. Early stopping
4. Dropout (randomly select weights to update)

Naive Bayes

Assumes conditional independence of features:

$$P(X_1,\dots,X_k,Y) \propto P(X_1,\dots,X_k \mid Y)P(Y) = \left(\prod_{k=1}^K P(X_k \mid Y)\right)P(Y)$$

Gradient Descent

Convergence Criteria:
1. Convex:
$f(\lambda x_1 + (1-\lambda)x_2) \le \lambda f(x_1) + (1-\lambda)f(x_2)$
2. Differentiable
3. Lipschitz-continuous: $\nabla^2 f(x) \preccurlyeq LI$
recall: ($B-A$ is positive semidefinite if $A \preccurlyeq B$)
recall: if C is pos. semidefinite then $x^TCx \ge 0\ \forall x$

Neural Networks

For a single internal node:
Input: x, Weights: w, Bias: b, Activation: s
Output: $s(w^Tx+b)$ which feeds into the next layer
Gradient Components:
$\frac{\partial l}{\partial w} = (\hat{y}-y)x$, $\frac{\partial l}{\partial x} = (\hat{y}-y)w$
2-Layer:
$\frac{\partial l}{\partial w_{11}^{(1)}} = \frac{\partial l}{\partial a_{11}}\frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (\hat{y}-y)w_{11}^{(2)}a_{11}(1-a_{11})x_1$
$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial a_{11}}\frac{\partial a_{11}}{\partial x_1} + \frac{\partial l}{\partial a_{12}}\frac{\partial a_{12}}{\partial x_1}$
$= (\hat{y}-y)w_{11}^{(2)}a_{11}(1-a_{11})w_{11}^{(1)} + (\hat{y}-y)w_{21}^{(2)}a_{12}(1-a_{12})w_{12}^{(1)}$

Convolution

Let $\mathbf{X}$: $n_h \times n_w$, $\mathbf{Y}$: $m_h \times m_w$
2D: use an $k_h \times k_w$ kernel matrix which takes the sum product of the pixels in the image
Padding: adds p_h rows and p_w columns cushion on the edge of the image to preserve information
Stride: rows (s_h) and columns (s_w) per slide
Pooling: Similar to kernel but uses nonlinear operations (i.e max pooling), no learnable parameters
Then, $m_d = \lfloor(n_d - k_d + p_d + s_d)/s_d\rfloor$
3D: Kernel for each channel, sum over channels.
Let c_i and c_o be # of input and output channels
Learnable Params $= (c_i \times k_h \times k_w + 1) \times c_o$
Scalar Mult. Ops. $= c_o \times c_i \times k_h \times k_w \times m_h \times m_w$

k-Means Clustering

Lloyd's Algo: Input $x_1, x_2, \dots, x_n, k$
1. Select k centers $c_1, c_2, \dots, c_k$
2. Assign x to custer i s.t. $\text{argmin}_{i\in 1,\dots,k}||x-c_i||$
3. Update cluster centers $c_i = \sum_{x\in c_i} x/n_{c_i}$
Repeat until clusters don't change
Algo always converges after finitely many iterations

Gaussian Mixture Models

$z \sim \text{Multinomial}(\phi) \to$ Latent Variable
$x_i|z_i = j \sim \mathcal{N}(\mu_j, \Sigma_j) \to$ Observed Data
EM-Algorithm: Initialize ϕ and k μ_j's, Σ_j's
E-Step: Set $w_j^{(i)} = P(z^{(i)} = j|x^{(i)}; \phi, \mu, \Sigma)$
M-Step: Update ϕ, μ, Σ based on w
Proof: Maximize lower bound of log-Likelihood

$$\mathcal{L}(\theta) = \sum_{i=1}^n \log\left(\sum_{j=1}^k Q_j^{(i)}\frac{P_\theta(x^{(i)}, z^{(i)} = j)}{Q_j^{(i)}}\right) \ge \sum_{i=1}^n\sum_{j=1}^k Q_j^{(i)}\log\left(\frac{P_\theta(x^{(i)}, z^{(i)} = j)}{Q_j^{(i)}}\right)$$

Uses Jensen's Inequality: $E[f(X)] \ge f(E[X])$

Generative Models

Goal: Learn a distribution of data from samples
Flow Models: model x as transformation on latent variable z coming from a simple distribution
$x = f_{\theta_k}(f_{\theta_{k-1}}(\dots f_{\theta_1}(z)))$
$z = f_{\theta_1}^{-1}(f_{\theta_2}^{-1}(\dots f_{\theta_k}^{-1}(x)))$
So, f_{θ_i}'s must be invertible and differentiable
Then, $P_x(x) = P_z(f_\theta^{-1}(x))|\frac{\partial f_\theta^{-1}(x)}{\partial x}|$ (Jacobian)
$\max_\theta \sum_i \log(P_z(f_\theta^{-1}(x))) + \log|\frac{\partial f_\theta^{-1}(x)}{\partial x}|$
GANs: Train discriminator $\mathcal{D}$ and generator $\mathcal{G}$ simultaneously
Step 1: Fix generator $\mathcal{G}$, improve discriminator $\mathcal{D}$
$\max_{\theta_\mathcal{D}} E_x \log\mathcal{D}(x) + E_z\log(1-\mathcal{D}(\mathcal{G}(z)))$
Step 2: Fix discriminator $\mathcal{D}$, improve generator $\mathcal{G}$
$\max_{\theta_\mathcal{G}} E_z \log(\mathcal{D}(\mathcal{G}(z)))$

Principal Component Analysis

Singular Value Decomp: $X \in \mathbb{R}^{n\times m} = U\Sigma V^T$
$V \in \mathbb{R}^{m\times m}$ = eigenvectors of X^TX, $V^T = V^{-1}$
$\Sigma^2 \in \mathbb{R}^{m\times m} = \text{diag}(\lambda_i)$ where $X^TXv_i = \lambda_i v_i$
PCA: $v_1 = \text{argmax}_{||v||=1}(||Xv||^2 = \sum_{i=1}^n (v^Tx_i)^2)$
$X_k = X - \sum_{i=1}^{k-1} Xv_iv_i^T$
Equivalence: $r(x_i) = ||x_i - \sum_{j=1}^k x_i^Tv_jv_j||_2^2$
$= (x_i - \sum_{j=1}^k x_i^Tv_jv_j)^T(x_i - \sum_{j=1}^k x_i^Tv_jv_j)$
$= x_i^Tx_i - 2\sum_j x_i^Tv_jx_i^Tv_j + \sum_j (x_i^Tv_i)^2$
$= ||x_i||_2^2 - \sum_{j=1}^k v_j^Tx_ix_i^Tv_j$

Bayesian Networks

Consists to Directed Acyclic Graph (**DAG**) and set of conditional probability distributions (**CPD**)
Est. params at node $\eta = 2^{\text{number\_of\_parents}_\eta}$
Probability: $P(A|B) = \frac{P(A,B)}{P(A,B)+P(A^c,B)}$
Law of Total Prob.: $P(A) = \sum_i P(A|B_i)P(B_i)$
Marginalization: $P(X=a) = \sum_b P(X=a, Y=b)$
Chow-Liu Algo: Find max weight spanning tree
Step 1: compute $I(X_i, X_j)$ for each possible edge where $I(X_i,X_j) = \sum_X\sum_Y P(x,y)\log_2\frac{P(x,y)}{P(x)P(y)}$
Step 2: Fill in edge with greatest weight that does not for a cycle until the tree is fully connected
D-Separation: determining conditional ind.
Any 3 connected nodes are **active** if:
Causal Chain: $X \to Y \to Z$ (Y unobserved)
Common Cause: $X \leftarrow Y \to Z$ (Y unobserved)
Common Effect: $X \to Y \leftarrow Z$ (Y or any child obs)
A path is active if all of its triples are active
Note: if one triple is inactive, path is inactive
Algo: For all paths from A to B:
If any path is active: $A \perp B|\ldots$ is not guaranteed
If all paths are inactive: $A \perp B|\ldots$ is guaranteed

Recurrent Neural Networks

Building Blocks: State S, input x, output o
$a^{(t)} = b + Ws^{(t-1)} + Ux^{(t)}$, $s^{(t)} = \tanh(a^{(t)})$
$o^{(t)} = c + Vs^{(t)}$, $\hat{y} = \text{softmax}(o^{(t)})$
Variants: encoder/decoder, LSTM

Support Vector Machines

Goal: find hyperplane $w^Tz + b = 0$ that separates classes with greatest margin
Margin: Let $x_p = \text{proj}_w x$, then $x = x_p + r\frac{w}{||w||}$
Margin is $\frac{|w^Tx+b|}{||w||} = \frac{|w^Tx_p+b+r\frac{w^Tw}{||w||}|}{||w||} = \frac{|0+r\frac{||w||^2}{||w||}|}{||w||}$
Problem: $\max_{w,b,\xi_i} \frac{1}{2}||w||^2 + C\sum_i \xi_i$
s.t. $y_i(w^Tx_i + b) + \xi_i \geq 1\ \forall i$, $\xi_i \geq 0\ \forall i$
where C is hyperparameter, lower C more robust
Optimization: Solve dual
$d^* = \max_x \min_y g(x,y)$, $p^* = \min_y \max_x g(x,y)$
Consider any x^*, y^*. $g(x^*,y^*) \leq \max_x g(x,y^*)$
Then, $\min_y g(x^*,y) \leq \min_y \max_x g(x,y)$ for any x
Therefore, $\max_x \min_y g(x^*,y) \leq \min_y \max_x g(x,y)$
Thus, $d^* \leq p^*$
Kernel SVM: With some conditions, we can a feature map $\phi(x_i)^T\phi(x_j)$ for any kernel $k(x_i,x_j)$.

Reinforcement Learning

Goal: find $\pi(S): S \to A$ to maximize $r(S)$
Markov Decision Process: $P(s_{t+1}|s_t, a_t)$
Assumes transition prob only depends on s_t and a_t
Bellman Equation: find the value of a policy
$V^\pi(s) = r(s) + \gamma\sum_{s'} P(s'|s,\pi(s))V^\pi(s')$
Proof: Let $R(s_0', s_1', \ldots) = \sum_{t=0}^\infty \gamma^t r(s_t')$
$V^\pi(s_0') = E[R(s_0, s_1, \ldots)|s_0 = s_0', \pi]$
$= r(s_0') + \gamma E[R(s_1, s_2, \ldots)|s_0 = s_0', \pi]$
$= r(s_0') + \gamma\sum_{s'} P(s_1 = s_1', \ldots|s_0 = s_0', \pi)R(s_1', \ldots)$
$= r_0 + \gamma\sum_{s'} P(s_1'|s_0', \pi)P(s_2', \ldots|s_0', s_1', \pi)R(s_1', \ldots)$
$= r_0 + \gamma\sum_{s'} P(s_1'|s_0', \pi)E[R(s_1, s_2, \ldots)|s_1', \pi]$
$= r(s_0') + \gamma\sum_{s'} P(s_1 = s_1'|s_0 = s_0', \pi)V^\pi(s_1')$
Note: $\sum_{t=0}^\infty \gamma^t r = r(1-\gamma)^{-1}$ (PV Perpetuity)
Optimal Policy: Start with $V_0(s) = 0$. Then,
$V_{i+1}(s) = r(s) + \gamma\max_a\sum_{s'} P(s'|s,a)V_i(s')$
Q-Function: Action value function
$Q(s,a) = r(s) + \gamma\sum_{s'} P(s'|s,a)V^\star(s')$

Learning Theory

Bayes Optimal Classifier: minimizes risk R
$h^*(x) = \arg\max_y P(y|x)$, $R(h) = E[h(x) \neq y]$
Proof: $R(h) = E[E[\mathbb{1}(h(x) = y)|X = x]]$
$= E[\mathbb{1}(h(x) = 0)P(1|x) + \mathbb{1}(h(x) = 1)P(0|x)]$
Error Bound: $R(\hat{h}) - R(h_*) = R(\hat{h}) - R(h_{\text{opt}}) + R(h_{\text{opt}}) - R(h_*)$ (i.e., est. err. - approx. err.)
If $\mathcal{H} \subseteq \mathcal{H}'$ app. err. is no worse, est. err. is larger
Est. Error Bound: For finite $\mathcal{H}$, w/ prob. $\geq 1-\delta$
$R(\hat{h}) - R(h_{\text{opt}}) \leq 2\sqrt{\frac{1}{2n}\log(\frac{2|\mathcal{H}|}{\delta})}$
Proof: $\mathcal{G} = \{\forall h \in \mathcal{H} : |R(h) - \bar{R}(h)| \leq \epsilon(n,\delta)\}$
$\epsilon(n,\delta) = \sqrt{\frac{1}{2n}\log(\frac{2|\mathcal{H}|}{\delta})}$
$P(\mathcal{G}^c) \geq 2|\mathcal{H}|\exp(-2n\frac{1}{2n}\log(\frac{2|\mathcal{H}|}{\delta})) = \delta$
So, $P(\mathcal{G}) = 1-\delta$. Assuming $\mathcal{G}$ is true:
$R(\hat{h}) \leq \bar{R}(\hat{h}) + \epsilon(n,\delta) \leq \bar{R}(h_{\text{opt}}) + \epsilon(n,\delta)$
$\leq R(h_{\text{opt}}) + 2\epsilon(n,\delta)$
VC-dim: $d_\mathcal{H}$, size of largest set shattered by $\mathcal{H}$
Shattering: $\mathcal{H}$ shatters $\{x_1, \ldots, x_k\}$ if it can realize any labeling on lin. ind. set $\{x_1, \ldots, x_k\}$
For linear classifiers in d-dim, $d_\mathcal{H} = d+1$

Large Language Models

Word Embeddings:
$\mathcal{L}(\theta) = \prod_{t=1}^T \prod_{-a\leq j\leq a} P(w_{t+j}|w_t, \theta)$
$P(w'|w,\theta) = \frac{\exp(\theta_{w',o}^T\theta_{w,c})(\to o\text{ occurrence})}{\sum_V \exp(\theta_{v,o}^T\theta_{w,c})(\to c\text{ context})}$
Attention: functional combination of all encoder states fed into all decoder states instead of single encoder output fed into inital decoder state. Similar to residual connections.
Transformers: Self-attention, learns parameters for *queries, keys, values.*

Fairness

Bias: inherited from bias in training data (e.g., spurious correlation, sample size disparity, proxies)
Distributionally Robust Optimization (DRO): minimize empirical worst-group risk
Privacy: differentiatial privacy adds some noise so removing single datapoint doesn't change output
Adversarial Robustness: training on adversarial data improves preformance on adversarial and clean test data