COMP 540 Final Note

1 Linear models for regression

1.1 Least squares regression

- Loss function: $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))^2$
  Vectorized form: $J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$
- Gradient: $\nabla_\theta J(\theta) = \frac{1}{m}X^T(X\theta - y)$
- Closed form solution: $\hat{\theta} = (X^TX)^{-1}X^Ty$
- Assume $y^{(i)} = (\theta^*)^Tx^{(i)} + \epsilon$, where $E[\epsilon] = 0, Var[\epsilon] = \sigma^2$: $E[\hat{\theta}] = \theta^*, Var[\hat{\theta}] = (X^TX)^{-1}\sigma^2$.
- $\equiv$ MLE on θ with normal distributed error ϵ.

1.2 L2-regularization: ridge regression

- Loss function:
  $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))^2 + \frac{\lambda}{2m}\sum_{j=1}^{d}\theta_j^2$
- Closed form solutionn: $\hat{\theta} = (X^TX + \lambda I)^{-1}X^Ty$
- $\equiv$ MAP on θ with prior: $\theta \sim \mathcal{N}(0, \alpha^2 I)$, with $\frac{\lambda}{m} = \frac{\sigma^2}{\alpha^2}$.

1.3 L1-regularization: lasso regression

- Loss function:
  $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))^2 + \frac{\lambda}{2m}\sum_{j=1}^{d}|\theta_j|$
- $\equiv$ MAP on θ with prior: $\theta_j \sim Laplace(0, \alpha), \forall j$

1.4 Locally weighted linear regression

- Loss function: $J(\theta) = \frac{1}{2m}\sum_{i=0}^{m}w^{(i)}(y^{(i)} - \theta^Tx^{(i)})^2$
  where $w^{(i)} = \exp\left(-\frac{(x-x^{(i)})^T(x-x^{(i)})}{2\sigma^2}\right)$
  Vectorized form: $J(\theta) = \frac{1}{2m}(X\theta - y)^TW(X\theta - y)$
- Gradient: $\nabla_\theta J(\theta) = \frac{1}{m}X^TW(X\theta - y)$
- Closed form solution: $\hat{\theta} = (X^TWX)^{-1}X^TWy$
- Non-parametric method.

2 Linear models for classification

2.1 Discriminative models for classification

- $P(y = 1|x) = h_\theta(x) = \frac{1}{1+\exp(-\theta^Tx)}$
  $\Rightarrow \log\frac{P(y=1|x)}{P(y=0|x)} = \theta^Tx$
- Loss (cross-entropy) function: (*convex & has a global minimum*) $J(\theta) =$
  $-\frac{1}{m}\sum_{i=0}^{m}y^{(i)}\log h_\theta(x^{(i)}) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))$
- L2-regularizion: $J_{reg}(\theta) = J(\theta) + \frac{\lambda}{m}\sum_{j=1}^{d}\theta_j^2$
  L1-regularizion: $J_{reg}(\theta) = J(\theta) + \frac{\lambda}{m}\sum_{j=1}^{d}|\theta_j|$

2.2 Generative models for classification

2.2.1 Gaussian discriminant analysis (GDA)

- Assumptions: $y \sim Bernoulli(\phi)$
  $x|y = 0 \sim \mathcal{N}(\mu_0, \Sigma)$ & $x|y = 1 \sim \mathcal{N}(\mu_1, \Sigma)$
- Likelihood: $\mathcal{L}(D) = \prod_{i=1}^{m}P(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma)$
  $= \prod_{i=1}^{m}\phi^{y^{(i)}}(1 - \phi)^{(1-y^{(i)})}\mathcal{N}(x^{(i)}|\mu_1, \Sigma)^{y^{(i)}}\mathcal{N}(x^{(i)}|\mu_0, \Sigma)^{(1-y^{(i)})}$
- Estimation for parameters:
  $\phi = \frac{1}{m}\sum_{i=1}^{m}y(i)$
  $\mu_1 = \frac{\sum_{i=1}^{m}y^{(i)}x^{(i)}}{\sum_{i=1}^{m}y^{(i)}}, \mu_0 = \frac{\sum_{i=1}^{m}(1-y^{(i)})x^{(i)}}{\sum_{i=1}^{m}(1-y^{(i)})}$
  $\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x - \mu_{y^{(i)}})(x - \mu_{y^{(i)}})^T$
- Linear decision boundaries when same Σ; quadratic boundaries when each class has its own Σ.

2.2.2 Naive Bayes models

- Assumptions: $P(x|y) = \prod_{j=1}^{d}P(x_j|y)$
- Bernoulli Naive Bayes models are estimated using counts, regularize using Beta prior ($\equiv$ pre-count).

2.3 Model criteria

- False negative: positive predicted to be negative.
- False positive: negative predicted to be positive.
- specificity $= P(y_{pred} = 0|y = 0) = \frac{TN}{FP+TN}$
- sensitivity $= P(y_{pred} = 1|y = 1) = \frac{TP}{FN+TP}$
- true positive rate (TPR) = sensitivity
- false positive rate (FPR) = 1 - specificity
- ROC curve represents FPR and TPR as a function of classification threshold. $0.5 \leq$ (area under curve) ≤ 1.0.

2.4 Multiclass classification

2.4.1 One vs. All (OVA) & One vs. One (OVO)

- OVA: not theoretically justified; simple and widely used.
- OVO: needs $O(K^2)$ classifiers for K classes; overfitting!

2.4.2 Softmax

- Log-likelyhood:
  $\ell(\mathcal{D}) = \frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{K}I(y^{(i)} = c)\log\frac{\exp(\theta^{(c)T}x^{(i)})}{\sum_{c'}\exp(\theta^{(c')T}x^{(i)})}$
- Regularized loss: $J(\theta) = -\ell(\mathcal{D}) + \frac{\lambda}{2m}\sum_{j=1}^{d}\sum_{c=1}^{K}\theta_j^{(c)2}$
- Gradient: $\nabla_\theta J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}x^{(i)}(I\{y^{(i)} = c\} - P(y^{(i)} = c|x^{(i)}; \theta)) + \frac{\lambda}{m}\sum_{j=1}^{d}\theta_j^{(c)}$

3 Kernel methods

3.1 Kernel functions

- Gaussian kernel: $\kappa(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$
- Polynomial kernel: $\kappa(x, x') = (1 + x^Tx')^p$
- Mercer's theorem ($\Leftrightarrow$ valid kernel): Gram matrix K whose elements are $\kappa(x^{(i)}, x^{(j)}), 1 \leq i, j \leq m$, should be positive definite for all possible $\{x^{(i)}|1 \leq i \leq m\}$
  $\Leftrightarrow \exists\phi$ s.t. $\kappa(x, x') = \phi(x)^T\phi(x')$

3.2 Perceptron

- Prediction: $h_\theta(x) = \text{sign}(\theta^Tx)$
- Update rule: $\theta \leftarrow \theta + \eta x^{(i)}y^{(i)}$, when $h_\theta(x^{(i)})y^{(i)} = -1$
- Convergence bounds: Let $\|x^{(i)}\| \leq R, \forall 1 \leq i \leq m$, the perceptron converges in at most $\frac{R^2\|\theta^*\|^2}{\gamma^2}$ updates, where $\gamma > 0$, $y^{(i)}(\theta^Tx^{(i)}) \geq \gamma, \forall 1 \leq i \leq m$. (*Appendix*)
- Kernalized version: (if $\eta = 1$, $\theta = \sum_{(x,y)\in\mathcal{D}_{mistake}}xy$)
  $\hat{y} = \text{sign}(\sum_{(x^{(i)},y^{(i)})\in\mathcal{D}}\alpha^{(i)}\langle x^{(i)}, x\rangle)$, in training: update $\alpha^{(i)} \leftarrow \alpha^{(i)} + y^{(i)}$, when $y\hat{y} = -1$

3.3 Support vector machine (SVM)

3.3.1 Maximize margin

- Optimization problem: $\min_{\theta,\theta_0}\frac{1}{2}\|\theta\|^2$, subject to $y^{(i)}(\theta^Tx^{(i)} + \theta_0) \geq 1, \forall 1 \leq i \leq m$
- Dual problem:
  $\max_\alpha\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_jy^{(i)}y^{(j)}\langle x^{(i)}, x^{(j)}\rangle$
  subject to: $\alpha_i \geq 0, \forall 1 \leq i \leq m$; $\sum_{i=1}^{m}\alpha_iy^{(i)} = 0$
- Results of solving Lagrange:
  $\theta = \sum_{i=1}^{m}\alpha^{(i)}x^{(i)}y^{(i)}$ & $\sum_{i=1}^{m}\alpha^{(i)}y^{(i)} = 0$
- KTT condition yields:
  $\alpha^{(i)}[y^{(i)}(\theta^Tx^{(i)} + \theta_0) - 1] = 0, \forall 1 \leq i \leq m$
- Prediction: $h_\theta(x) = \text{sign}(\sum_{i=1}^{m}\alpha^{(i)}y^{(i)}\langle x^{(i)}, x\rangle + \theta_0)$

3.3.2 Non-separable case

- Optimization problem: $\min_{\theta,\theta_0}\frac{1}{2}\|\theta\|^2 + C\sum_{i=1}^{m}\xi^{(i)}$, subject to $y^{(i)}(\theta^Tx^{(i)} + \theta_0) \geq 1 - \xi^{(i)}, \forall 1 \leq i \leq m$; and $\xi^{(i)} \geq 0, \forall 1 \leq i \leq m$
- Dual problem:
  $\max_\alpha\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_jy^{(i)}y^{(j)}\langle x^{(i)}, x^{(j)}\rangle$
  subject to: $0 \leq \alpha_i \leq C, \forall 1 \leq i \leq m$; $\sum_{i=1}^{m}\alpha_iy^{(i)} = 0$

3.3.3 Hinge loss – *another view of SVM*

- Define $h_\theta(x) = \theta^Tx + \theta_0$, rewrite constraints into $\xi^{(i)} = \max(0, 1 - y^{(i)}h_\theta(x))$.
- Loss function:
  $J(\theta) = C\sum_{i=1}^{m}\max(0, 1 - y^{(i)}h_\theta(x)) + \frac{1}{2}\|\theta\|^2$

3.3.4 Multiclass SVM

- Loss function:
  $J(\theta) = \sum_{i=1}^{m}\sum_{j\neq y^{(i)}}\max\left(0, \theta^{(j)T}x^{(i)} - \theta^{y^{(j)}T}x^{(i)} + \Delta\right)$
- Δ is some fixed margin.

3.3.5 SVM for regression

- Loss function: $L_\epsilon(y, \hat{y}) = \max(0, |y - \hat{y}| - \epsilon)$ in
  $J(\theta) = C\sum_{i=1}^{m}L_\epsilon(y^{(i)}, \theta^Tx^{(i)} + \theta_0) + \frac{1}{2}\theta^T\theta$
- $J(\theta)$ is convex but not differentiable, the optimization problem is unconstrained.
- Quadratic programming (QP) problem:
  $\min_\theta C\sum_{i=1}^{m}(\xi_+^{(i)} + \xi_-^{(i)}) + \frac{1}{2}\theta^T\theta$, subject to
  $\xi_+^{(i)} \geq 0, \xi_-^{(i)} \geq 0, \forall 1 \leq i \leq m$;
  $\theta^Tx^{(i)} + \theta_0 - \xi_-^{(i)} - \epsilon \leq y^{(i)} \leq \theta^Tx^{(i)} + \theta_0 + \xi_+^{(i)} + \epsilon$
- Lagrange yields: $\theta = \sum_{i=1}^{m}(\alpha_+^{(i)} - \alpha_-^{(i)})x^{(i)}$, where $\alpha_+^{(i)}$ and $\alpha_-^{(i)}$ are Lagrange multipliers of two constraints.
- Prediction: $h_\theta(x) = \sum_{i=1}^{m}(\alpha_+^{(i)} - \alpha_-^{(i)})\langle x^{(i)}, x\rangle + \theta_0$

4 Neural Networks

Note: A feed forword network with a linear output layer and at least one hidden layer with any squashing activation function (e.g. sigmoid, ReLU), can approaximate any function from $\mathbb{R}^d \to \mathbb{R}$ to arbitrary percision with enough hidden units.

4.1 Activation functions

- $sigmoid(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$
  $\frac{\mathrm{d}}{\mathrm{d}x}\sigma(x) = \sigma(x)(1-\sigma(x))$
- $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ (*unbias, better than sigmoid*)
- $\mathrm{ReLU}(x) = \max(0, x)$
- Softmax function: $g(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$ (*output layer*)

4.2 Backpropagation

- Using chain rule, propagate derivatives in inverse order.
- Gradients need to be added up at forks (*accumulative*)!

4.3 Convolutional neural networks

- Let K be filter number, F be filter size, S be stride, P be padding.
- A conv layer takes as input of a volume $W_1 \times H_1 \times D_1$, produces an output volume $W_2 \times H_2 \times D_2$:
  $W_2 = \frac{W_1-F+2P}{S}+1$
  $H_2 = \frac{H_1-F+2P}{S}+1$
  $D_2 = K$
- Total number of parameters: $F \times F \times D_1$ weights per filter; $F \times F \times D_1 \times K$ weights, K biases.
- Pooling layer (subsampling): input $W_1 \times H_1 \times D$, output $W_2 \times H_2 \times D$:
  $W_2 = \frac{W_1-F}{S}+1$
  $H_2 = \frac{H_1-F}{S}+1$
- Classical architecture: [(CONV – RELU)*N – POOL]*M – (FC – RELU)*K – SOFTMAX

4.4 Optimization for training deep models

4.4.1 Momentum

- **Require**: learning rate α, initial parameter θ, batch size m, momentum parameter μ, initial velocity v.
- Sample a mini batch of m examples $(x^{(i)}, y^{(i)})$
- Compute gradient estimate $\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(h_\theta(x^{(i)}), y^{(i)})$
- Compute velocity update $v \leftarrow \mu v - \alpha\hat{g}$
- Apply update $\theta \leftarrow \theta + v$

4.4.2 Nesterov momentum

- **Require**: learning rate α, initial parameter θ, batch size m, momentum parameter μ, initial velocity v.
- Sample a mini batch of m examples $(x^{(i)}, y^{(i)})$
- Apply interim update $\tilde{\theta} \leftarrow \tilde{\theta} + \mu v$
- Compute gradient at interim point
  $\hat{g} = \frac{1}{m}\nabla_{\tilde{\theta}}\sum_i L(h_{\tilde{\theta}(x^{(i)}),y^{(i)}})$
- Compute velocity update $v \leftarrow \mu v - \alpha\hat{g}$
- Apply update $\theta \leftarrow \theta + v$

4.4.3 AdaGrad

- **Require**: step size α, initial parameter θ, batch size m, $\delta = 10^{-7}$ (constant for numerical stability)
- Compute gradient on minibatch
  $\hat{g}_{ij}^{(l)} \leftarrow \frac{1}{m}\nabla_{\theta_{ij}}^{(l)}\sum_i L(h_\theta(x^{(i)}), y^{(i)})$
- Accumulate squared gradient $r_{ij}^{(l)} \leftarrow r_{ij}^{(l)} + \hat{g}_{ij}^{(l)} * \hat{g}_{ij}^{(l)}$
- Compute gradient $\Delta\theta_{ij}^{(l)} \leftarrow -\frac{\alpha}{\delta+\sqrt{r_{ij}^{(l)}}}\hat{g}_{ij}^{(l)}$
- Apply update $\theta_{ij}^{(l)} \leftarrow \theta_{ij}^{(l)} + \Delta\theta_{ij}^{(l)}$

4.4.4 RMSprop

- **Require**: step size α, initial parameter θ, batch size m, $\delta = 10^{-7}$ (constant for numerical stability), exponential decay rate ρ.
- Compute gradient on minibatch
  $\hat{g}_{ij}^{(l)} \leftarrow \frac{1}{m}\nabla_{\theta_{ij}}^{(l)}\sum_i L(h_\theta(x^{(i)}), y^{(i)})$
- Accumulate squared gradient $r_{ij}^{(l)} \leftarrow \rho r_{ij}^{(l)} + (1-\rho)\hat{g}_{ij}^{(l)} * \hat{g}_{ij}^{(l)}$
- Compute gradient $\Delta\theta_{ij}^{(l)} \leftarrow -\frac{\alpha}{\delta+\sqrt{r_{ij}^{(l)}}}\hat{g}_{ij}^{(l)}$
- Apply update $\theta_{ij}^{(l)} \leftarrow \theta_{ij}^{(l)} + \Delta\theta_{ij}^{(l)}$

4.4.5 Adam (adaptive moments)

- **Require**: step size α (10^{-3} default), initial parameter θ, batch size m, exponential decay rates for moment estimates ρ_1 (0.99 default) & ρ_2 (0.999 default), $\delta = 10^{-7}$ (constant for numerical stability)
- Initialize time step $t = 0$, first and second moment $s = 0$, $r = 0$
- Compute gradient on minibatch
  $\hat{g}_{ij}^{(l)} \leftarrow \frac{1}{m}\nabla_{\theta_{ij}}^{(l)}\sum_i L(h_\theta(x^{(i)}), y^{(i)})$
- $t \leftarrow t+1$
- Update biased first moment $s_{ij}^{(l)} \leftarrow \rho_1 s_{ij}^{(l)} + (1-\rho_1)\hat{g}_{ij}^{(l)}$
- Update biased second moment
  $r_{ij}^{(l)} \leftarrow \rho_2 r_{ij}^{(l)} + (1-\rho_2)\hat{g}_{ij}^{(l)} * \hat{g}_{ij}^{(l)}$
- Correct bias in first moment $\hat{s}_{ij}^{(l)} = \frac{s_{ij}^{(l)}}{1-\rho_1^t}$
- Correct bias in second moment $\hat{r}_{ij}^{(l)} = \frac{r_{ij}^{(l)}}{1-\rho_2^t}$
- Compute gradient $\Delta\theta_{ij}^{(l)} \leftarrow -\frac{\alpha\hat{s}_{ij}^{(l)}}{\delta+\sqrt{\hat{r}_{ij}^{(l)}}}$
- Apply update $\theta_{ij}^{(l)} \leftarrow \theta_{ij}^{(l)} + \Delta\theta_{ij}^{(l)}$

5 Decision trees

5.1 Cost functions

- Misclassification rate: ($\hat{y}$ = the majority label in $\mathcal{D}$)
  $cost(\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{(x,y)\in\mathcal{D}} I(y \neq \hat{y})$
- Entropy: (p = fraction of positive examples in $\mathcal{D}$)
  $cost(\mathcal{D}) = -p\log_2 p - (1-p)\log_2(1-p)$
- Gini index: (same as entropy)
  $cost(\mathcal{D}) = 2p(1-p)$

5.2 Decision tree for regression

- Define $cost(\mathcal{D}) = \sum_{i=1}^m (y^{(i)} - \overline{y})^2$
  where $\overline{y} = \frac{1}{|\mathcal{D}|}\sum_{i=1}^m y^{(i)}$

5.3 Node is not worth splitting when

- Node is pure
- Depth exceeds max depth
- $|\mathcal{D}_{\text{left}}|$ or $|\mathcal{D}_{\text{right}}|$ is too small
- Reduction in cost is too small

5.4 Avoid overfitting

- Early stopping – stop growing the tree when the decrease in error is not sufficient to justify the complexity of an additional level.
- Post pruning – grow the full tree and then prune using a validation set to guide subtree removal. Evaluate CV error on each subtree and pick tree whose error is within 1 standard deviation of minimum.

6 Ensembles

6.1 Bagging

- Assume errors of individual members are uncorrelated.
- For regression: when $h_l(x) = f(x) + \epsilon_l(x)$ for $1 \leq l \leq L$, and $\epsilon_l \sim \mathcal{N}(0, \sigma_l{}^2)$
  $\Rightarrow E_{bag} = \frac{1}{L}E_{avg}$ (*expected squared error*)
- For classification: ϵ = error of each classifier, and $\epsilon < \frac{1}{2}$
  $\Rightarrow E_{bag} = \sum_{i=\frac{L}{2}+1}^{L}\binom{i}{L}\epsilon^i(1-\epsilon)^{L-i}$

6.2 Boosting

- Loss function: $J_l = \sum_{i=1}^m w_l^{(i)}\mathrm{I}(h_l(x^{(i)}) \neq y^{(i)})$
- Prediction: $h(x) = \mathrm{sign}(\sum_{l=1}^L \alpha_l h_l(x))$
- Adaboost algorithm: initialize $w_1^{(i)} = \frac{1}{m}$, for $1 \leq i \leq m$
  1. fit h_l to minimize $J_l = \frac{1}{m}\sum_{i=1}^m w_l^{(i)} L(y^{(i)}, h_l(x^{(i)}))$
  2. calculate error rate $\epsilon_l = \frac{\sum_{i=1}^m w_l^{(i)}\mathrm{I}(h_l(x^{(i)} \neq y^{(i)}))}{\sum_{i=1}^m w_l^{(i)}}$
  3. calculate $\alpha_l = \frac{1}{2}\log\left(\frac{1-\epsilon_l}{\epsilon_l}\right)$, stop if $\epsilon_l \geq \frac{1}{2}$
  4. update $w_{l+1}^{(i)} = \begin{cases} w_l^{(i)}\exp(\alpha_l), & \text{incorrect on } x^{(i)} \\ w_l^{(i)}\exp(-\alpha_l), & \text{correct on } x^{(i)} \end{cases}$

6.3 Gradient boosting

6.3.1 Gradient boosting for regression

- Residuals are negative gradients (squared error loss):
  $J = \sum_{i=1}^m \frac{1}{2}(y - h(x))^2 \Rightarrow \frac{\partial J}{\partial h(x^{(i)})} = h(x^{(i)}) - y^{(i)}$

6.3.1 Gradient boosting for classification

- Loss function: $J = \frac{1}{m}\sum_{i=1}^m D_{KL}(y^{(i)}, h(x^{(i)}))$
- Gradient boosting algorithm (k classes):
  – Start with an initial $h^0 \ldots h^k$ for $x^{(1)} \ldots x^{(m)}$
  – Repeat until convergence: Calculate matrix of gradients, it each h_{add} to the negative gradient, $h \leftarrow h + h_{add}$

7 Probabilistic graphical models

7.1 Directed models – Bayesian network

- $P(X) = \prod_i P(x_i | Parents(x_i))$
- Reduce number of parameters $O(k^n) \rightarrow O(nk^m)$ if each variable in graph has no more than m parents.

7.2 Undirected models – Markov network

- $\tilde{P}(X) = \prod_{C\in G}\phi(C)$ (C is a clique in graph)
- Partition function: $Z = \int_X \tilde{P}(X)\mathrm{d}X$ or $Z = \sum_X \tilde{P}(X)$
- $P(X) = \frac{1}{Z}\tilde{P}(X)$
- Energy function E: $\tilde{P}(X) = \exp(-E(X))$
  high (low) energy $\Leftrightarrow$ low (high) $\tilde{P}(X)$

7.3 Sampling

7.3.1 Ancestral sampling

- For directed graphical models, polynomial time.
- Algorithm:
  – Sort variables in topological order
  – Sample x_i from distribution $P(x_i|Parents(x_i))$

7.3.2 Gibbs sampling

- For undirected graphical models.
- Algorithm:
  – Start with randomly generated values $x_1, ..., x_n$
  – Iteratively visit each x_i and sample a value for it based on $P(x_i|Neighbors(x_i))$
  – Repeat previous step, generate stream of samples

7.4 Hidden Markov models

- Specified by sets S (hidden states), O (observations) and probability parameters $\lambda = [\pi, a, b]$
  – π is initial state probability
  – a is hidden state transition probability
  – b is emission probability
- Inference problems:
  – Filtering: $P(X_t|e_1, ..., e_t)$
  – Smoothing: $P(X_k|e_1, ..., e_t), k < t$
  – Most likely state sequence:
  $\arg\max_{X_1,...,X_t} P(X_1, ..., X_t|e_1, ..., e_t)$

7.4.1 Forward computation – filtering

- Define: $\alpha_t(i) = P(e_1, ..., e_t, X_t = s_i)$
- Algorithm:
  – $\alpha_0(i) = \pi_i$, $1 \leq i \leq n$ where $|S| = n$
  – $\alpha_{t+1}(i) = b_j(e_{t+1}) \sum_{i=1}^{n} \alpha_t(i) a_{ij}$, $1 \leq j \leq n$, $0 \leq t \leq T-1$
- Time complexity: $O(n^2 T)$

7.4.2 Backward computation – smoothing

- $P(X_k|e_1, ..., e_t) \propto P(e_{k+1}, ..., e_t|X_k) P(X_k|e_1, ..., e_t)$ where $t > k$
- Define: $\beta_k(i) = P(e_{k+1}, ..., e_t|X_k = s_i)$
- Algorithm:
  – $\beta_T(i) = 1$, $1 \leq i \leq n$
  – $\beta_k(i) = \sum_{j=1}^{n} a_{ij} b_j(e_{k+1}) \beta_{k+1}(j)$, $1 \leq j \leq n$, $0 \leq k \leq T-1$
- Time complexity: $O(n^2 T)$

7.4.3 Viterbi algorithm – most likely sequence

- Define:
  $\delta_t(i) = \max_{X_1,...,X_{t-1}} P(X_1, ..., X_{t-1}, X_t = s_i, e_1, ..., e_t)$
- Algorithm:
  – $\delta_0(i) = \pi(i)$, $1 \leq i \leq n$
  – $\delta_{t+1}(j) = \max_i \delta_t(i) a_{ij} b_j(e_{t+1})$, $1 \leq j \leq n$, $0 \leq t \leq T-1$
- Time complexity: $O(n^2 T)$

7.4.4 Parameter estimation

- Paired sequences: $\hat{a}_{ij} = \frac{\# s_i \to s_j}{\# s_i}$ & $\hat{b}_j(e_k) = \frac{\# s_j \to e_k}{\# s_j}$
- Observation sequences only – Baum-Welch EM:
  – Define: $\xi_t(i, j) = P(X_t = s_i, X_{t+1} = s_j|e_1, ..., e_T, \lambda)$
  $\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(e_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_t(i) a_{ij} b_j(e_{t+1}) \beta_{t+1}(j)}$
  – Let $\gamma_t(i) = P(X_t = s_i|e_1, ..., e_T, \lambda) = \sum_{j=1}^{n} \xi_t(i, j)$
  – Estimate $\hat{\pi}_i = \gamma_1(i)$, $\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$,
  $\hat{b}_j(e_k) = \frac{\sum_{t=1}^{T} \gamma_t(j) * I(E_t = e_k)}{\sum_{t=1}^{T} \gamma_t(j)}$
  – Algorithm:
    – Guess $\lambda_0 = [\pi_0, a_0, b_0]$
    – Repeat until convergence:
      – Calculate α, β from λ
      – Re-estimate λ from α, β

8 Unsupervised learning

8.1 Principal components analysis (PCA)

- Assume: data distrubution is unimodal Gaussian (fully explained by mean & variance)
- Assume: information to be preserved is in the variance
- Project data $\mathbb{R}^d \to \mathbb{R}^k$ ($k < d$), maximizing variance.

8.1.1 PCA method

- Zero-mean, unit variance transform on $\mathcal{D}$
- Find S = covariance matrix of transformed $\mathcal{D}$
- Find $\lambda_1, ..., \lambda_k$ (the k largest eigenvalues of S) and associated eigenvectors $u_1, ..., u_k$
- Project $x^{(i)} \mapsto [u_1^T x^{(i)}, ..., u_k^T x^{(i)}]^T$, where $x^{(i)} \in \mathbb{R}^d$

8.1.2 Kernel PCA

- Idea: map $x^{(i)} \mapsto \phi(x^{(i)})$ where $\phi : \mathbb{R}^d \to \mathbb{R}^D$ ($D >> d$)
- The result when project back to $\mathbb{R}^d$ will be nonlinear.
- Algorithm:
  – Pick a kernel
  – Construct kernel matrix K over data $x^{(1)}, ..., x^{(m)}$
  – Zero mean the kernel matrix K to get $\tilde{K}$
  – Solve the eigenvalue problem $\tilde{K}\alpha = \lambda\alpha$, $\alpha \in \mathbb{R}^m$
  – For a new x, we project it as: $y_j = \sum_{i=1}^{m} \alpha_j^{(i)} \kappa(x, x^{(i)})$, for $j = 1, ..., L$ (# of components) where eigenvectors are ordered by value

8.2 Expectation maximization algorithm (EM)

8.2.1 K-means

- Cost function: $J = \sum_{i=1}^{m} \sum_{k=1}^{K} z_k^{(i)} \|x^{(i)} - \mu_k\|^2$
- E-step: cluster assignment, minimize J wrt. z, fix μ
- M-step: relocate means, minimize J wrt. μ, fix z
- Time complexity: $O(mK)$ per iteration
- Converges to local minimum & vulnerable to outliers.

8.2.2 Gaussian mixture model (GMM)

- Generative mode:
  $P(x^{(i)}) = \sum_{k=1}^{K} P(z^{(i)} = k) P(x^{(i)}|z^{(i)} = k)$
  $z^{(i)} \sim \text{Multinomial}(\pi); \pi_k > 0, \sum_k \pi_k = 1$
  $x^{(i)}|_{z^{(i)}=k} \sim \mathcal{N}(\mu_k, \Sigma_k)$
- Infer $z^{(i)}$ for each $x^{(i)}$, where $\theta = \{\pi, \mu, \Sigma\}$:
  $P(z^{(i)} = k|x^{(i)}; \theta) = \frac{P(z^{(i)}=k) P(x^{(i)}|z^{(i)}=k;\theta)}{\sum_{k'} P(z^{(i)}=k') P(x^{(i)}|z^{(i)}=k';\theta)}$
- Soft EM algorithm:
  – Guess values of $\theta = \{\pi, \mu, \Sigma\}$
  – E-step: calculate the responsibility of each component toward generating $x^{(i)}$: $r_k^{(i)} = P(z^{(i)} = k|x^{(i)}; \theta)$
  – M-step: given $r_k^{(i)}$ and $x^{(i)}$, $1 \leq i \leq m$, $1 \leq k \leq K$, re-estimate π, μ, Σ:
  $\pi_k = \frac{1}{m} \sum_{i=1}^{m} r_k^{(i)}$, $\mu_k = \frac{\sum_{i=1}^{m} r_k^{(i)} x^{(i)}}{\sum_{i=1}^{m} r_k^{(i)}}$,
  $\Sigma_k = \frac{\sum_{i=1}^{m} r_k^{(i)} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T}{\sum_{i=1}^{m} r_k^{(i)}}$

9 Reinforcement learning

9.1 Markov decision process (MDP)

9.1.1 The model

- A set of states S, a subset of which are terminal states.
- Actions(s): possible actions from state s, no actions from terminal states.
- A transition function $T(s, a, s')$: probability of transitioning to state s' if action a is taken in state s.
- A reward function $r(s, a, s')$: reward for taking action a in state s and ending up in state s', or $r(s, a)$ or $r(s)$.

9.1.2 The value function – *expected utility*

- Definition: $V_\pi(s) = E\left[\sum_{t=0}^{\infty} r(s_t, \pi(s_t))|\pi, s_0 = s\right]$
- Recursive:
  $V_\pi(s) = \sum_{s'} T(s, \pi(s), s') \left[r(s, \pi(s), s') + V_\pi(s')\right]$
- Q-function: $Q_\pi(s, a) = \sum_{s'} T(s, a, s') \left[r(s, a, s') + V_\pi(s')\right]$

9.1.3 Policy evaluation

- Approach 1: set up linear equations and solve for V_π
- Approach 2: (iterative improvement) $O(|S|)$ each iteration
  – Repeat until convergence:
    – For each state s:
      –
  $V_\pi^i(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left[r(s, \pi(s), s') + V_\pi^{i-1}(s')\right]$

9.1.4 Optimality

- Definition: $V*(s) = V_{\pi^*}(s) = \max_\pi V_\pi(s)$
- Bellman's equation:
  $V^*(s) = \max_{a \in \text{Actions}(s)} T(s, a, s') \left[r(s, a, s') + V^*(s')\right]$
- Given V^*: $Q^*(s, a) = \sum_{s'} T(s, a, s') \left[r(s, a, s') + V^*(s)\right]$
- Given Q^*: $V^*(s) = \max_{a \in \text{Action}(s)} Q^*(s, a)$

9.2 Solving MDPs

9.2.1 Policy iteration

- Policy improvement:
  – Compute $Q_\pi(s, a)$ from $V_\pi(s)$
  – Update π: $\pi'(s) = \arg\max_{a \in \text{Action}(s)} Q(s, a)$
- Policy iteration algorithm:
  – Start with a random policy π
  – Repeat until no change to policy occurs:
    – Compute value of policy π (policy evaluation)
    – Improve the policy at each state (policy improvement)

9.2.2 Value iteration

- Note: no explicit policy.
- Value iteration algorithm:
  - Start with $V^{(0)}(s) = 0$ for all states s in S
  - Repeat until convergence:
    - Bellman update: $V^i(s) \leftarrow \max_{a \in \text{Action}(s)} \sum_{s'} T(s,a,s') \left[r(s,a,s') + V^{i-1}(s') \right]$

9.3 Model-based RL

9.4 Model-free RL

9.5 TD learning

9.6 Q learning

Appendix

Distributions

- Poisson distribution PMF: $P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$
- Normal distribution PDF: $f(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$
- Multivariate normal distribution PDF: $f(X;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} \exp\left(-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)\right)$
- Laplace distribution PDF: $f(x;\mu,b) = \frac{1}{2b}\exp\left\{-\frac{|x-\mu|}{b}\right\}$
- Beta distribution PDF: $f(x;\alpha,\beta) = \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}$
- Given that $X_1 \sim Pois(\lambda_1)$, $X_2 \sim Pois(\lambda_2)$, $X = X_1 + X_2$, X_1 and X_2 are independent: $X \sim Pois(\lambda_1 + \lambda_2)$.
- Given that $P(X_0 = x_0) = \alpha_0 \exp\left\{-\frac{(x_0-\mu_0)^2}{2\sigma_0^2}\right\}$, and that $P(X_1 = x_1|X_0 = x_0) = \alpha_1 \exp\left\{-\frac{(x_1-x_0)^2}{2\sigma_1^2}\right\}$: $P(X_1 = x_1) = \alpha_0\alpha_1\sqrt{\frac{2\pi\sigma_0^2\sigma_1^2}{\sigma_0^2+\sigma_1^2}} \exp\left\{-\frac{1}{2}\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_1^2}\right\}$

Information theory

- Conditional information: $H(Y|X) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\frac{p(x)}{p(x,y)}$
- Mutual information: $I(X;Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\frac{p(x,y)}{p(x)P(y)}$
- Kullback-Leibler (KL) divergence: $D_{KL}(p||q) = \sum_x p(x)\log\frac{p(x)}{q(x)}$
- Cross entropy: $H(p,q) = E_p[-\log q] = H(p) + D_{KL}(p||q)$

Convex function

- A function $f(x)$ is convex on a set S iff for $\lambda \in [0,1]$, and $\forall x,y \in S$: $f(\lambda x + (1-\lambda)y) \le \lambda f(x) + (1-\lambda)f(y)$.
- A function $f(x)$ is convex on a set S iff $\frac{\mathrm{d}^2}{\mathrm{d}x^2}f(x)$ is positive semidefinite everywhere in the set.

Convergence of perceptron

Proof. Let $\theta^{(k-1)}$ be the parameter vector when the algorithm makes a mistake on (x,y).

$$\theta^{(k)} = \theta^{(k-1)} + \eta xy$$

Take dot product on both sides with θ^* (some separating hyperplane)

$$\begin{aligned}\theta^{*T}\theta^{(k)} &= \theta^{*T}(\theta^{(k-1)} + \eta xy) \\ &= \theta^{*T}\theta^{(k-1)} + \eta y(\theta^{*T}x) \\ &\ge \theta^{*T}\theta^{(k-1)} + \eta\gamma\end{aligned}$$

If $\theta^{(0)}$ is all zeros vector, then

$$\theta^{*T}\theta^{(k)} \ge \eta k\gamma \tag{1}$$

From the update rule,

$$\begin{aligned}\theta^{(k)} &= \theta^{(k-1)} + \eta xy \\ \|\theta^{(k)}\|^2 &= \|\theta^{(k-1)} + \eta xy\|^2 \\ &= \|\theta^{(k-1)}\|^2 + \eta^2y^2\|x\|^2 + 2\eta y(\theta^{(k)T}x) \\ &\le \|\theta^{(k-1)}\|^2 + \eta^2\|x\|^2 \\ &\le \|\theta^{(k-1)}\|^2 + \eta^2R^2\end{aligned}$$

Starting with $\theta^{(0)}$ of all zeros,

$$\|\theta^{(k-1)}\|^2 \le k\eta^2R^2 \tag{2}$$

Putting (1) and (2) together, $k \le \frac{R^2\|\theta^*\|^2}{\gamma^2}$. □

Mercer's theorem

Proof. Since K is positive definite, $K = u^T\Lambda u$, where Λ is a diagonal matrix with entries $\lambda^{(i)} > 0$. Consider an element $\kappa(x^{(i)}, x^{(j)})$ of K. We can construct this element as follows

$$\kappa(x^{(i)}, x^{(j)}) = (\Lambda^{\frac{1}{2}}u_{:,i})^T(\Lambda^{\frac{1}{2}}u_{:,j})$$

Now define $\phi(x^{(i)}) = \Lambda^{\frac{1}{2}}u_{:,i}$, then we have $\kappa(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T\phi(x^{(j)})$. □

Adaboost principle

Kernel PCA derivation

Correctness of EM

Proof. For any $\mathbf{Z}$ with non-zero probability $p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta})$, we can write:

$$\log p(\mathbf{X}|\boldsymbol{\theta}) = \log p(\mathbf{X},\mathbf{Z}|\boldsymbol{\theta}) - \log p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta})$$

We take the expectation over possible values of the unknown data $\mathbf{Z}$ under the current parameter estimate $\boldsymbol{\theta}^{(t)}$ by multiplying both sides by $p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}^{(t)})$ and summing over $\mathbf{Z}$. The left-hand side is the expectation of a constant, so we get:

$$\begin{aligned}\log p(\mathbf{X}|\boldsymbol{\theta}) = &\sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}^{(t)})\log p(\mathbf{X},\mathbf{Z}|\boldsymbol{\theta}) \\ &- \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}^{(t)})\log p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}) \\ =&Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) + H(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})\end{aligned}$$

where $H(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$ is defined by the negated sum it is replacing. This last equation holds for any value of $\boldsymbol{\theta}$ including $\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)}$,

$$\log p(\mathbf{X}|\boldsymbol{\theta}^{(t)}) = Q(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)}) + H(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)})$$

and subtracting this last equation from the previous equation gives

$$\begin{aligned}\log p(\mathbf{X}|\boldsymbol{\theta}) - \log p(\mathbf{X}|\boldsymbol{\theta}^{(t)}) =&Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) - Q(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)}) \\ &+ H(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) - H(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)})\end{aligned}$$

However, Gibbs' inequality tells us that $H(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) \ge H(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)})$, so we can conclude that

$$\log p(\mathbf{X}|\boldsymbol{\theta}) - \log p(\mathbf{X}|\boldsymbol{\theta}^{(t)}) \ge Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) - Q(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)})$$

In words, choosing $\boldsymbol{\theta}$ to improve $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$ beyond $Q(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)})$ cannot cause $\log p(\mathbf{X}|\boldsymbol{\theta})$ to decrease below $\log p(\mathbf{X}|\boldsymbol{\theta}^{(t)})$, and so the marginal likelihood of the data is non-decreasing. □