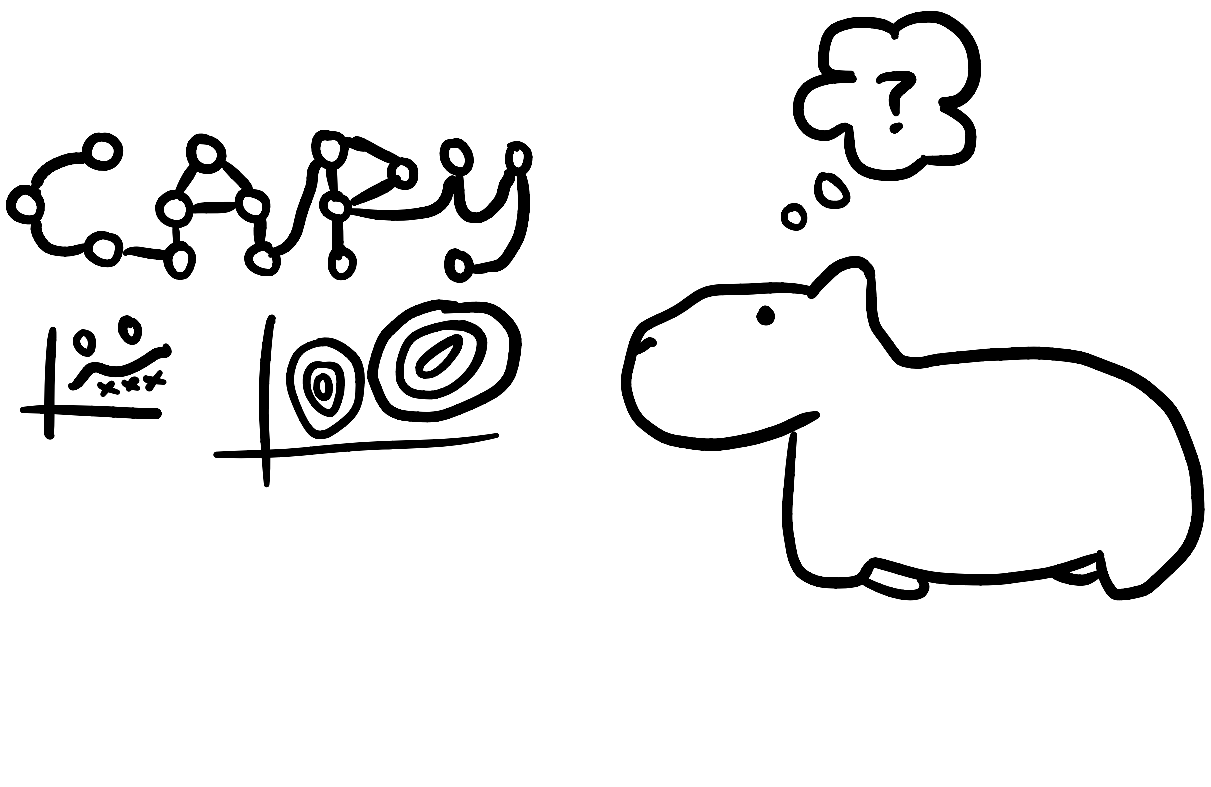
CMU 10-701 F21

NOTES

wanshenl



Lecture 08/30 Intro

- Logistics
  - Ziv Bar-Joseph, Patrick Virtue
  - Piazza for everything
  - Midterm 10/27 Wednesday 6pm
- Machine Learning
  - Methods that can help generalize information from observed data so that it can be used to make better decisions in the future
  - Contrast
    - Statistics: understanding data at hand
    - Artificial Intelligence: build an intelligent agent
    - Data Mining: patterns from large scale data
- 30+ minutes of sales pitch

- Three axes
  - Data – this class assumes you have (possibly incomplete) data
  - Algorithms
    - Model-based: probabilistic, parametric, non-parametric

$$\begin{tikzcd} \text{Data} \arrow[r, "\text{model learning}"] & \text{Model} \arrow[r, "\text{model inference}"] & \text{Knowledge} \end{tikzcd}$$

      - Learning: from data to model
        - Model = summary of data = inform on data generation process
      - Inference: from model to knowledge
        - Given model, answer questions
    - Model-free
  - Tasks (later)

- Parametric models
  - Fixed-size models that do not grow with the data
  - More data = learn/fit model better
- Non-parametric models
  - Models that grow with the data
  - More data = more complex models
- Discriminative models
  - Find best line separating some set of points
  - No generative assumption

- Tasks
  - Prediction: estimate output given input
    - Classification (discrete labels), Regression (continuous labels)
  - Description: (unsupervised learning) no supervision in data as to descriptive outputs
    - Density estimation, Clustering, Embedding
  - Formally
    - Supervised learning: given $D=\{X_i, Y_i\}$ learn model $F: X_k \rightarrow Y_k$
    - Unsupervised learning: given $D=\{X_i\}$ group data into Y classes using model $F: X_i \rightarrow Y_i$
    - Reinforcement learning: given $D=\{$environment, actions, rewards$\}$ learn policy $F_1: \{e, r\} \rightarrow a$ and reward $F_2: \{a, e\} \rightarrow R$
    - Active learning: given $D=\{X_i, Y_i\}, \{X_j\}$ learn $F_1: \{X_j\} \rightarrow x_k$ maximizing success of supervised learning $F_2: \{X_i, x_k\} \rightarrow Y$

Lecture 09/01 Probability, MLE, MAP

- Probability
  - Random variable : event whose status is unknown
  - Domain Ω : the set of values a random variable can take
    - Binary / Discrete / Continuous
  - Axioms
    1. $0 \leq P(A) \leq 1$
    2. $P(\text{true}) = 1$, $P(\text{false}) = 0$
    3. $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
  - Prior : degree of belief in an event in the absence of any other information
  - Conditional probability : $P(A=1 \mid B=1)$ is the fraction of cases where A is true if B is true
  - Joint distributions $P(A \cap B)$ or $P(A,B)$: probability that a set of random variables will take a specific value
    - If we assume independence, $P(A,B) = P(A)\,P(B)$
  - Chain rule : holds for any pair of random variables
    - $P(A,B) = P(A \mid B)\,P(B)$
  - Bayes rule

$$P(A \mid B) = \frac{P(B \mid A)\,P(A)}{P(B)} = \frac{P(B \mid A)\,P(A)}{\sum_A P(B \mid A)\,P(A)}$$

  - Bayes rule for conditional distributions

$$f(x \mid y) = \frac{f(y \mid x)\,f(x)}{f(y)} = \frac{f(y \mid x)\,f(x)}{\int f(y \mid x)\,f(x)\,dx}$$

- Conditional distributions
  - A statistical model is a collection of distributions, the parameters specify individual distributions
  - Normal (Gaussian) distributions $x \sim N(\mu, \sigma^2)$ $\quad \underbrace{\theta = (\mu, \sigma^2)}_{\text{parameters}}$

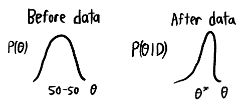
$$p(x \mid \theta) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

  - Multi-variate Gaussian

$$P(X \mid \theta) = \frac{1}{(2\pi)^{n/2}\,|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right]$$

- Density estimation
  - Bayesian learning

$$P(\theta \mid D) = \frac{P(D \mid \theta)\,P(\theta)}{P(D)}$$



  - But note the denominator doesn't really matter

$$\underset{\text{posterior}}{P(\theta \mid D)} \propto \underset{\text{likelihood}}{P(D \mid \theta)}\ \underset{\text{prior}}{P(\theta)}$$

- Prior distribution
  - Source of prior?
    - Expert knowledge
    - Simple posterior form
    - Uniform (uninformative) prior
    - Conjugate priors (next page)

Lecture 09/01 cont.

- **Conjugate prior**
  - Closed form representation of prior
  - $P(q)$ and $P(q|D)$ have the same form as a function of θ
    - Coin flip example

      Bernoulli Likelihood $\quad P(D|\theta) = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$

      If prior is Beta $\quad P(\theta) = \dfrac{\theta^{\beta_H-1}(1-\theta)^{\beta_T-1}}{B(\beta_H,\beta_T)} \sim \text{Beta}(\beta_H,\beta_T)$

      Then posterior is Beta $\quad P(\theta|D) \sim \text{Beta}(\beta_H+\alpha_H, \beta_T+\alpha_T)$

    - Dice roll example

      Multinomial Likelihood $\quad P(D|\theta) = \theta_1^{\alpha_1}\theta_2^{\alpha_2}\theta_3^{\alpha_3}\cdots\theta_k^{\alpha_k}$

      If prior is Dirichlet $\quad P(\theta) = \dfrac{\prod_{i=1}^{k}\theta_i^{\beta_i-1}}{B(\beta_1,\dots,\beta_k)} \sim \text{Dirichlet}(\beta_1,\dots,\beta_k)$

      Then posterior is Dirichlet $\quad P(\theta|D) \sim \text{Dirichlet}(\beta_1+\alpha_1,\dots,\beta_k+\alpha_k)$

- Posterior distribution
  - Approach above is **Bayesian**
    - Prior encoded as distribution over parameter values
    - Bayes rule for updated posterior distribution

- **MAP (Maximum A Posteriori) Estimation**
  - Choose θ that maximizes a posterior probability

    $$\hat{\theta}_{MAP} = \underset{\theta}{\text{argmax}}\ P(\theta|D) = \underset{\theta}{\text{argmax}}\ P(D|\theta)P(\theta)$$

  - Not widely used in practice, needs to **assume prior**

- Density Estimation
  - Learn mapping from set of attributes to probability
  - Binary/discrete variables: just count
  - Continuous variables: fit a model
  - Trivial learning algorithm for discrete variables

    $$\hat{P}(x_i=u) = \frac{\#\text{ records in which } x=u}{\text{total } \#\text{ of records}}$$

- **Maximum Likelihood Principle**

  $$\hat{P}(\text{dataset}|\underset{\uparrow\ \text{model}}{M}) = \hat{P}(x_1\wedge x_2\wedge\cdots\wedge x_n|M) = \prod_{k=1}^{n}\hat{P}(x_k|M)$$

  - For coin flip $\quad P(D|M) = q^{n_1}(1-q)^{n_2}$

    $$\underset{q}{\text{argmax}}\ P(D|M) \Rightarrow q = \frac{n_1}{n_1+n_2}$$

Lecture 09/01 cont.

- Log Probabilities
  - Sometimes probabilities are too small, use logspace instead to remove exponents etc

$$\log \hat{P}(\text{dataset}|M) = \log \prod_{k=1}^{n} \hat{P}(x_k|M) = \sum_{k=1}^{n} \log \hat{P}(x_k|M)$$

$\uparrow$ since log monotonic between 0 and 1, maximizing this also maximizes original

- Maximum Likelihood Principle
  - Fit statistical models by maximizing probability of generating observed samples

$$L(x_1, \dots, x_n | \theta) \underset{\substack{\uparrow \\ \text{assume samples} \\ \text{are independent}}}{=} p(x_1|\theta) \cdots p(x_n|\theta)$$

  - For the Gaussian

$$\bar{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{\sigma^2} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{\mu})^2$$

- MLE vs MAP
  - MLE: choose value that maximizes probability of observed data

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}}\ P(D|\theta)$$

  - MAP: choose value that is most probable given observed data and prior belief

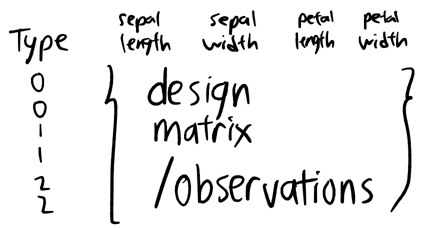
$$\hat{\theta}_{MAP} = \underset{\theta}{\operatorname{argmax}}\ P(\theta|D) = \underset{\theta}{\operatorname{argmax}}\ P(D|\theta)P(\theta)$$

Lecture 09/08 Decision Theory

- Logistics
  - HW1 out, due Wed 9/22 2359
- Fisher Iris Dataset 1936
  - $D = \{(y^{(i)}, \vec{x}^{(i)})\}_{i=1}^{N} = \{(y^{(i)}, x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)})\}_{i=1}^{N}$

    $y \in \{0,1,2\}^N$

    $X \in \mathbb{R}^{N \times 4}$ design matrix



- Density estimation review
  - $D = \{x_2^{(i)}\}_{i=1}^{N}$
  - Assume Gaussian $X_2 \sim \text{Normal}(\mu, \sigma^2)$
  - $p(D;\theta) = p(x^{(1)}, x^{(2)}, x^{(3)}; \mu^{(1)}, \sigma^{(1)}, \mu^{(2)}, \sigma^{(2)}, \mu^{(3)}, \sigma^{(3)})$ expand notation

    $= p(x^{(1)}, x^{(2)}, x^{(3)}; \mu, \sigma)$ identically distributed

    $= p(x^{(1)}; \mu, \sigma)\, p(x^{(2)}; \mu, \sigma)\, p(x^{(3)}; \mu, \sigma)$ independent

    $= \prod_{i=1}^{3} p(x^{(i)}; \mu, \sigma)$
  - MLE
    - $p(D;\theta) = \prod_{i=1}^{N} p(x_2^{(i)}; \mu, \sigma)$
    - Likelihood $L(\theta; D) = p(D;\theta)$
    - $\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ L(\theta; D)$

Lecture 09/08 cont.

- Density estimation $D = \{\vec{x}^{(i)}\}_{i=1}^{N}$ $D \to \hat{\theta}$
- Supervised learning $D = \{y^{(i)}, \vec{x}^{(i)}\}_{i=1}^{N}$ $h(\vec{x}) \to \hat{y}$

- Iris data supervised learning
  - Classification error rate $\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(y^{(i)} \neq \hat{y}^{(i)})$

- Bayes decision rule

```
def h(x):
    if $p(x|Y=1)p(Y=1) \geq p(x|Y=0)p(Y=0)$:
        return 1
    else:
        return 0
```

$$P(y=i|X) = \frac{P(X|y=i)\,P(y=i)}{P(X)} \overset{def}{=} \underset{\uparrow \text{class}}{q_i(x)}$$

- Optimal classification function
  - $h^*(x) = \underset{y}{\text{argmax}}\ P(Y=y|X=x)$
  - Goal: minimize expected loss for random test data (X,Y)
    $h^* = \underset{h}{\text{argmin}}\ \mathbb{E}_{XY}[L(Y, h(x))]$
  - Loss function $L: Y \times Y \to \mathbb{R}$
    - Two class 0-1 loss
    - Two class arbitrary loss
  - Risk = expected loss
    - $R(h) = \mathbb{E}_{XY}[L(Y, h(x))]$

Lecture 09/13 kNN, Naive Bayes

- Classifiers : three major groups
  1. Instance-based : use observations directly, no models, e.g., kNN
  2. Generative : generative statistical model, e.g., Bayesian network
  3. Discriminative : directly estimate a decision rule/boundary, e.g., decision tree

- k Nearest Neighbor kNN
  - Select class based on majority vote in the k closest points
  - Need to define distance function
  - How to find a good value of k?
    - Cross validation, later
    - Influences smoothness of classifier
      - Almost like a kernel method, but depends on input data not on parameters

- Naive Bayes classifier
  - For Bayes decision rule above, how to compute $P(\underset{\text{joint on all input}}{X}|\underset{\text{class label}}{y})$?
  - Suppose 16 binary attributes $\to \{0,1\}$ class. How many parameters needed for fully determining $P(X|y)$?
    - $2^{16}-1$ parameters for each class $\Rightarrow$ infeasible

Lecture 09/13 cont.

- Naive Bayes classifier cont.
  - Assume given the class label, attributes are conditionally independent
    $P(X|y) = \prod_j P_j(x^j|y)$ where p_j = model for attribute j
  - Now you only need 16 parameters total for previous example
  - Hence

$$\hat{y} = \text{argmax}_v \; P(y=v|X)$$
$$= \text{argmax}_v \frac{P(X|y=v)\,P(y=v)}{P(X)}$$
$$= \text{argmax}_v \prod_j p_j(x^j|y=v)\,P(y=v)$$

- Conditional Likelihood

$$L(\underset{\uparrow\,\text{sample } i}{X_i} \,|\, y_i=1, \underset{\uparrow\,\text{set of all parameters}}{\theta}) = \prod_j p(x_i^j|y_i=1, \underset{\uparrow\,\text{specific parameters for attribute } j \text{ in class } 1}{\theta_i^j})$$

- Feature Transform
  - For text, Bag of Words is common
  - Document = collection of words encoded as a vector
  - Vector can be binary (present/absent) or discrete (# appearances)
  - Example: document X_i modeled by vector of m^* indicator features $\{\phi^j(X_i)\}$
    where $\phi^j(X_i)=1$ if word $j \in$ document X_i and is 0 otherwise.
    Then $\Phi(x_j) = [\phi^1(x_j) \cdots \phi^m(x_j)]^T$ is the feature vector for document X_i
    Also written $\Phi(x_j) = [\phi^1\,\phi^2 \cdots \phi^m]$.
  - English typically uses ~10k words.

- Problems with Naive Bayes
  - Assumption of conditional independence given class label is often violated
  - If insufficient data, not observed by training data, your probability can zero out
    - Pseudocounts: add one sample with all words, one sample with no words

- Naive Bayes classifier for continuous values
  - Usually Gaussian model is used, i.e., $X \sim N(\mu, \Sigma)$
    - In more depth,
      $y_i \sim \text{Multinomial}(p_1, p_2, \ldots, p_{N_y})$ generates output
      $x_j \sim N(\mu_i, \Sigma_i)$ generates corresponding input
    - To determine class,

$$P(X|y) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)\right] \rightarrow \text{need lots of data to compute mean } \mu, \text{ covariance } \Sigma$$

    - By applying Naive Bayes assumption, covariance Σ becomes diagonal matrix, only need to learn $x^j \sim N(\mu_v^j, \sigma_v^j)$

$$P(X|y=v) = \prod_j P(x^j|y=v) = \prod_j \frac{1}{(2\pi)^{1/2}\sigma_v^j} \exp\left[-\frac{(x^j-\mu_v^j)^2}{2(\sigma_v^j)^2}\right] \rightarrow \text{each class has its own mean and variance}$$

Lecture 09/15 Decision Trees

- Decision Trees
  - Internal nodes = attributes
  - Leafs = classification
  - Edges = assignment

- def buildTree(n, A): n=samples, A=attributes

```
if empty(A) or all n(L) same:
    status = leaf
    class = most common class in n(L)
else:
    status = internal
    a = bestAttribute(n, A)
    left = buildTree($n_{a=1}$, $A \setminus \{a\}$)
    right = buildTree($n_{a=0}$, $A \setminus \{a\}$)
```

- Entropy
  $H(X) = \Sigma_c -P(X=c) \log_2 P(X=c)$
- Conditional entropy
  $H(Y|X) = \Sigma_i P(X=i) H(Y|X=i)$
- Information gain
  $IG(Y|X) = H(Y) - H(Y|X)$
  Note $IG(Y|X) \geq 0$ by Jensen's

- bestAttribute = attribute that maximizes information gain at each node

- Avoiding overfitting
  - Tree pruning
    - Split data into train and test
    - Build tree with train
      - For all internal nodes starting at root,
        - Remove subtree rooted at node
        - Assign class to be most common among training
        - Check test data error
          - If error lower, keep change
          - Else restore subtree, repeat for all nodes in subtree

- Continuous values
  - Threshold to turn into binary / discretize

- Decision trees surprisingly very effective in practice

Lecture 09/20 Ensemble: bagging, random forest. Linear regression.

- Bagging / bootstrap aggregation
  - Reduce variance of an estimated prediction function
  - Classification: a committee of trees each cast a vote for the predicted class
  - Idea: randomly draw datasets with replacement from the training data, each sample same size
  - $Z = \{(x_1,y_1),(x_2,y_2),\dots,(x_N,Y_N)\}$
  - Z^{*b} where $b=1$ to B

    $\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$ ← prediction at input x when bootstrap sample b is used for training
  - Recommended >50% samples

- Random Forest
  - Extension to bagging: use a subset of the features (at each tree node!) instead of the samples
  - At each node choosing split feature, choose only among $m<M$ feature
  - Helps to create smaller decision trees
  - Recommended $\sqrt{n}$ features

- Linear Regression
  - Given x, predict y, where we assume $y=wx+\varepsilon$

- Least squares error
  - $\text{argmin}_w \sum_i (y_i - wx_i)^2$
  - Minimizes squared distance between measurements, predicted line

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

    $= 0$ when $w = \frac{\sum_i x_i y_i}{\sum_i x_i^2}$

- Adding a bias
  - If line doesn't pass through origin
  - $y = w_0 + w_1 x + \varepsilon$
  - $w_0 = \frac{1}{n}\sum_i (y_i - w_1 x_i)$ $w_1 = \frac{\sum_i x_i(y_i - w_0)}{\sum_i x_i^2}$

- Multiple inputs / multivariate regression
  - $y = w_0 + w_1 x_1 + \dots + w_k x_k + \varepsilon$

- Non-linear basis functions
  - As long as coefficients are linear, still a linear regression problem
  - Examples
    - Polynomial $\phi_j(x) = x^j$ for $j=0$ to n
    - Gaussian $\phi_j(x) = \frac{x-\mu_j}{2\sigma_j^2}$
    - Sigmoid $\phi_j(x) = \frac{1}{1+\exp(-s_j x)}$

- General linear regression
  - $y = \sum_{j=0}^{n} w_j \phi_j(x)$

Lecture 09/20 cont.

- General linear regression loss

  $J(w) = \Sigma_i \left(y^i - \Sigma_j w_j \phi_j(x^i)\right)^2$

  $J(w) = \Sigma_i \left(y^i - w^T \phi(x^i)\right)^2$

  $\frac{\partial}{\partial w} J(w) = 2\, \Sigma_i \left(w^T \phi(x^i) - y^i\right) \phi(x^i)^T$

  $= 0$ when $\Sigma_i y^i \phi(x^i)^T = w^T \left[\Sigma_i \phi(x^i) \phi(x^i)^T\right]$

  Let $\Phi = \begin{pmatrix} \phi_0(x^1) & \cdots & \phi_k(x^1) \\ \vdots & & \vdots \\ \phi_0(x^n) & \cdots & \phi_k(x^n) \end{pmatrix}$ n by k+1

  Then $w = (\Phi^T \Phi)^{-1} \Phi^T y$

  which is known as the pseudo inverse

- Probabilistic interpretation
  - $y = w^T \phi(x) + \varepsilon$
  - Then $w_{MLE} = (\Phi^T \Phi)^{-1} \Phi^T y$

- Extensions to linear regression
  - Note parameters learnt were global
  - Extensions adjust parameters based on input region

- Splines
  - Before: fit one function for entire region
  - Now: fit a set of piecewise (usually cubic polynomials satisfying continuity and smoothness constraints.
  - Need to define regions in advance, usually uniform

- Local Average Regression

- Local Kernel Regression

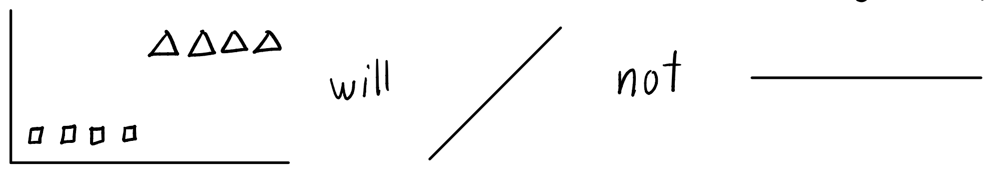
- Nadaraya-Watson Kernel Regression

- Spatially adaptive regression

Lecture 09/22 Logistic Regression

- Generative vs discriminative
  - Generative: rely on all points, compute P(X|Y)
  - Discriminative: care about boundary, do not compute P(X|Y)

- Linear regression
  - Problem: objective fn cares more about min dist than correctly classifying



Lecture 09/22 cont.

- Sigmoid function
  $$g(h) = \frac{1}{1+e^{-h}}$$

- Sigmoid binary classification
  - $p(y=0|X;\theta) = g(w^TX) = \frac{1}{1+e^{w^Tx}}$ note - can be absorbed into w
  - $p(y=1|X;\theta) = 1-g(w^TX) = \frac{1}{1+e^{-w^Tx}}$

- Sigmoid likelihood
  - $L(y|X;w) = \Pi_i (1-g(X_i;w))^{y_i} g(X_i;w)^{(1-y_i)}$
  - $\log L(y|X;w) = \Sigma_{i=1}^N y_i w^T X_i - \ln(1+e^{w^TX_i})$
  - $\frac{\partial}{\partial w_j}[\log L(y|X;w)] = \Sigma_{i=1}^N X_i^j(y_i - P(y^i=1|X_i;w))$

  helpful to write $g(X;w)$ and $1-g(X;w)$ in derivations
    - No closed form
    - But concave $\Rightarrow$ gradient ascent

- Gradient ascent
  - $w^j \leftarrow w^j + \varepsilon \Sigma_{i=1}^N X_i^j(y_i-(1-g(X_i;w)))$ where ε = small constant = learning rate

- Logistic regression
  1. Choose ε
  2. Start with guess for w
  3. For all j, $w^j \leftarrow w^j + \varepsilon \Sigma_{i=1}^N X_i^j (y_i - (1-g(X_i;w)))$
  4. Check if $LL(y|X;w) = \Sigma_{i=1}^N y_i \ln(1-g(X_i;w)) + (1-y_i)\ln(g(X_i;w))$
     - improved $\rightarrow$ go to step ③
     - same $\rightarrow$ stop

- Regularization
  - What if not enough data?
  - Regularize: impose additional constraints on the parameters are fitting
  - Add the prior: $p(y=1,\theta|X) \propto p(y=1|X;\theta)p(\theta)$
  - This changes the LL, e.g., with Gaussian prior,
    - $LL(y;w|X) = \Sigma_{i=1}^N y_i w^T X_i - \ln(1+e^{w^TX_i}) - \Sigma_j \frac{(w^j)^2}{2\sigma^2}$ assuming mean 0
    - $\underset{\text{MAP estimate}}{w^j} \leftarrow w^j + \varepsilon \Sigma_{i=1}^N [X_i^j(y_i-(1-g(X_i;w)))] - \varepsilon \frac{w^j}{\underset{\text{variance of prior model}}{\sigma^2}}$
  - Gaussian $\rightarrow$ L2 regularization, min w^2
  - L1 also popular, min $|w|$

- Multiclass logistic regression
  - For $i<k$, set $p(y=i|X;\theta) = g(w_i^0 + w_i^1 x^1 + \dots + w_i^d x^d) = g(w_i^T X)$

    where $g(z_i) = \frac{e^{z_i}}{1+\Sigma_{j=1}^{k-1} e^{z_j}}$ where $z_i = w_i^0 + w_i^1 x^1 + \dots + w_i^d x^d$

    and $P(y=k|X;\theta) = \frac{1}{1+\Sigma_{j=1}^{k-1} e^{z_j}}$

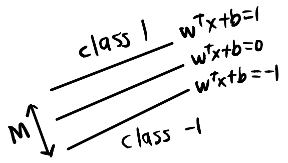
    and $w_m^j \leftarrow w_m^j + \varepsilon \Sigma_{i=1}^N X_i^j(\delta_m(y_i) - P(y_i=m|X_i;w))$ where $\delta_m(y_i)=1$ if $y_i=m$, 0 otherwise
  - Can also incorporate ϕ transforms of data

Lecture 09/27 Support Vector Machines

- SVM
  - Mainly works for 2 classes, extensions are heuristic (still works though)
  - Discriminative, not probabilistic

- Max margin classifier aka linear SVM
  - Instead of fitting all points, fit boundary points
  - Learn boundary maximizing margin from both sets of points
    - Margin = distance to closest point on either side
  - No proof, but works well in practice

- Specifying max margin classifier
  - $+1$ if $w^Tx+b \geq 1$
  - -1 if $w^Tx+b \leq -1$
  - Undefined if $-1 < w^Tx+b < 1$



Note: strong linear assumption

- Maximizing the margin
  - Observe: w orthogonal to $+1$ plane, -1 plane
  - Observe: if x^+ a point on $+1$ plane and x^- closest point to x^+ on the -1 plane then $x^+ = \lambda w + x^-$
  - Hence
    - $w^Tx^+ + b = +1$
    - $w^Tx^- + b = -1$
    - $x^+ = \lambda w + x^-$
    - $|x^+ + x^-| = M$

    $$\left.\right\} \text{Solving, } \lambda = \frac{2}{w^Tw}, \quad M = \frac{2}{\sqrt{w^Tw}}$$
  - So our solution should
    - Correctly classify all points
    - Minimize w^Tw

- Quadratic programming

$$\min_u \frac{u^TRu}{2} + d^Tu + c$$

subject to n inequality constraints

$$a_{11}u_1 + a_{12}u_2 + \cdots \leq b_1$$
$$\vdots$$
$$a_{n1}u_1 + a_{n2}u_2 + \cdots \leq b_n$$

and k equivalency constraints

$$a_{n+1,1}u_1 + a_{n+1,2}u_2 + \cdots = b_{n+1}$$
$$\vdots$$
$$a_{n+k,1}u_1 + a_{n+k,2}u_2 + \cdots = b_{n+k}$$

where
- u is unknown vector
- R is square matrix
- d is vector
- c is scalar

- QP problems have better solvers than gradient descent/simulated annealing

Lecture 09/27 cont.

- SVM as QP

  Last column in R is 0

  $\min (w^Tw)/2$

  subject to

  $\forall x$ in class $+1$, $w^Tx+b \geq 1$ ⎫ n constraints
  $\forall x$ in class -1, $w^Tx+b \leq -1$ ⎭

- Non linearly separable cases, how to QP?
  - Minimize training errors
    - $\min w^Tw$, $\min$ # errors
    - Hard to solve 2 minimization problems
  - Penalize training errors
    - $\min w^Tw + C$ (# errors)
    - Hard to encode in QP problem
  - Minimize distance between misclassified points and correct plane

    $\min_w (w^Tw)/2 + \sum_{i=1}^{n} C\varepsilon_i$

    subject to

    $\forall x_i$ in class $+1$, $w^Tx+b \geq 1-\varepsilon_i$ ⎫ n constraints
    $\forall x_i$ in class -1, $w^Tx+b \leq -1+\varepsilon_i$ ⎭

    $\forall i$, $\varepsilon_i \geq 0$ } n constraints

- Dual representation of SVM QP
  - This representation allows for a trick for easier math, faster runtime

- Lagrange multiplier

  $\min_x \; x^2$ st $x \geq b$ $\longrightarrow$ $\min_x \max_\alpha \; x^2 - \alpha(x-b)$ st $\alpha \geq 0$

- Lagrange multiplier for SVM
  - Linearly separable case

    Original

    $\min (w^Tw)/2$

    st $(w^Tx_i+b)y_i \geq 1$

    Dual

    $\min_{w,b} \max_\alpha \; (w^Tw)/2 - \sum_i \alpha_i [(w^Tx_i+b)y_i - 1]$

    st $\alpha_i \geq 0 \;\; \forall i$

    Taking derivatives, solves to

    $w = \sum_i \alpha_i x_i y_i$

    $b = y_i - w^Tx_i$ for i st $\alpha_i > 0$

    $\sum_i \alpha_i y_i = 0$

$\frac{\partial L}{\partial w} = w - \sum_i \alpha_i x_i y_i$

$= 0$ when $w = \sum_i \alpha_i x_i y_i$

$\frac{\partial L}{\partial b} = -\sum_i \alpha_i y_i$

$= 0$ when $\sum_i \alpha_i y_i = 0$

$\frac{\partial L}{\partial \alpha_i} = -[(w^Tx_i+b)y_i - 1] = 0$ when

$(w^Tx_i+b)y_i = 1$

$\Leftrightarrow (w^Tx_i+b)y_i^2 = y_i$

$\Leftrightarrow (w^Tx_i+b) = y_i$

(recall $y_i = \{1,-1\}$)

$\Rightarrow b = y_i - w^Tx_i + b$ for all i where $\alpha_i > 0$

Lecture 09/28 cont.

- In general, N data points are separable in a space of N-1 dimensions or more
  - Map original input space into higher dimensional feature space
  - High dimensionality and many more parameters are possible problems
  - But SVM well-suited: kernel tricks for efficient computation, dual formulation only assigns parameters to samples not features

- Quadratic kernel trick

$$\phi(x)\,\phi(z) = \begin{pmatrix} 1 \\ \sqrt{2}x^1 \\ \vdots \\ \sqrt{2}x^m \\ (x^1)^2 \\ \vdots \\ (x^m)^2 \\ \sqrt{2}x^1x^2 \\ \vdots \\ \sqrt{2}x^{m-1}x^m \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \sqrt{2}z^1 \\ \vdots \\ \sqrt{2}z^m \\ (z^1)^2 \\ \vdots \\ (z^m)^2 \\ \sqrt{2}z^1z^2 \\ \vdots \\ \sqrt{2}z^{m-1}z^m \end{pmatrix} \begin{matrix} \left.\right\} m+1 \text{ linear terms} \\ \left.\right\} m \text{ quadratic terms} \\ \left.\right\} \frac{m(m-1)}{2} \text{ pairwise terms} \end{matrix} = \underbrace{\sum_i 2x^iz^i}_{m} + \underbrace{\sum_i (x^i)^2(z^i)^2}_{m} + \underbrace{\sum_i \sum_{j=i+1} 2x^ix^jz^iz^j}_{m(m-1)/2} = 1$$

$$\sim m^2$$

- But note that

$$(\langle x,z\rangle+1)^2 = \langle x,z\rangle^2 + 2\langle x,z\rangle + 1$$
$$= \left(\sum_i x^iz^i\right)^2 + \left(\sum_i 2x^iz^i\right) + 1$$
$$= \left(\sum_i 2x^iz^i\right) + \left(\sum_i (x^i)^2(z^i)^2\right) + \left(\sum_i \sum_{j=i+1} 2x^ix^jz^iz^j\right) + 1$$

$=$

which only costs m operations

Lecture 10/04 Neural Networks

- Linear classifiers
  - Decision stumps
  - Generative model
  - Logistic regression
  - SVM
  - Perceptron
  - But for nonlinear data?
    - Feature mapping
    - Learn the boundary

- Perceptron alg w/o bias

```
t=1
$w_1 = \vec{0}$
On mistake:
    mistake positive    $w_{t+1} \leftarrow w_t + x$
    mistake negative    $w_{t+1} \leftarrow w_t - x$
```

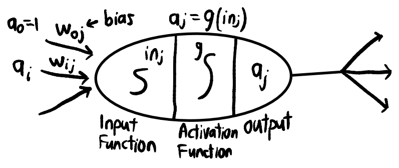
Classification

$h(x) = \text{sign}(w^Tx+b)$

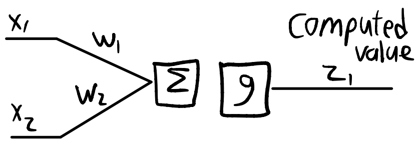
- Multilayer Perceptrons
  - Feedforward neural network with at least one hidden layer
  - Hidden layer has nodes that are neither inputs nor outputs

Lecture 10/4 cont.

- 1943 Neuron McCulloch & Pitts



- Single Neuron system
  - g step ⇒ Perceptron
  - g sigmoid ⇒ logistic regression
  - g identity ⇒ linear regression

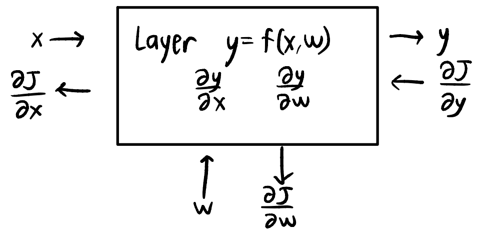


- Activation functions
  - Hard threshold $g(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$ $\frac{\partial g}{\partial z} = \begin{cases} 0 & z \geq 0 \\ 0 & z < 0 \end{cases}$
  - Sigmoid /softmax $g(z) = \frac{1}{1+\exp(-z)}$ $\frac{\partial g}{\partial z} = g(z)(1-g(z))$
  - ReLU $g(z) = \max(0, z)$ $\frac{\partial g}{\partial z} = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$

- Optimizing
  To find the best set of weights
  $\ell(y, \hat{y}) = (y - \hat{y})^2$
  $J^i(w) = \ell(y_i; h_w(x^i)) = (y^i - h_w(x^i))^2$
  $w \leftarrow w - \alpha \nabla_w J(w)$
  $\hat{y} = h_w(x) = g\left(\Sigma_j w_j x_j\right)$

- Loss functions
  - Regression has squared error $\ell(y, \hat{y}) = (y - \hat{y})^2$
  - Classification has cross entropy $\ell(y, \hat{y}) = -\Sigma_k y_k \log \hat{y}_k$

- Backpropagation
  - Compute derivatives per layer, forwards then backwards



$x \rightarrow$
$\frac{\partial J}{\partial x} \leftarrow$
Layer $y = f(x, w)$
$\frac{\partial y}{\partial x}$
$\frac{\partial y}{\partial w}$
$\rightarrow y$
$\leftarrow \frac{\partial J}{\partial y}$
w
$\frac{\partial J}{\partial w}$

Lecture 10/6 More Neural Networks

- Prove $\frac{\partial}{\partial v} v^T A v = (A^T + A)v$, $v \in \mathbb{R}^2$, $A \in \mathbb{R}^{2\times 2}$

  $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

  $Av = v_1\begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix} + v_2\begin{pmatrix} a_{12} \\ a_{22} \end{pmatrix} = \begin{pmatrix} v_1 a_{11} + v_2 a_{12} \\ v_1 a_{21} + v_2 a_{22} \end{pmatrix}$

  $v^T(Av) = v_1^2 a_{11} + v_1 v_2 a_{12} + v_1 v_2 a_{21} + v_2^2 a_{22}$

  $\frac{\partial}{\partial v_1} v^T A v = 2v_1 a_{11} + v_2 a_{12} + v_2 a_{21}$

  $\frac{\partial}{\partial v_2} v^T A v = 2v_2 a_{22} + v_1 a_{12} + v_1 a_{21}$

  $\therefore \frac{\partial}{\partial v} v^T A v = \begin{pmatrix} 2a_{11}v_1 + v_2 a_{12} + v_2 a_{21} \\ 2a_{22} + v_1 a_{12} + v_1 a_{21} \end{pmatrix}$

  $= \begin{pmatrix} 2a_{11} & a_{12}+a_{21} \\ a_{12}+a_{21} & 2a_{22} \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$

- Multivariate Chain Rule

  $y = f(\vec{z})$ $\quad$ $\vec{y} = f(\vec{z})$

  $\vec{z} = g(x)$ $\quad$ $\vec{z} = g(\vec{x})$

  $\frac{dy}{dx} = \sum_j \frac{\partial y_i}{\partial z_j}\frac{\partial z_j}{dx}$ $\quad$ $\frac{dy_i}{dx_k} = \sum_j \frac{\partial y_i}{\partial z_j}\frac{\partial z_j}{\partial x_k}$

- Practical neural network considerations
  - Large number of neurons ⇒ danger of overfitting
  - Modeling assumptions vs data assumptions
  - Gradient descent can easily get stuck in local optima (not convex)
  - If there are no non-linear activations, all linear layers can be reduced into single linear layer

- Universal approximation thm
  - A two-layer neural network with sufficiently many neurons can approximate any continuous function to any desired accuracy

- Tensorflow playground is nice

- Convolutional Neural Networks
  - History
    - 2005 HoG Histogram of oriented gradients
    - HoG → linear filter
    - AlexNet breakthrough
  - Convolution
    - Signal processing

      $z[i,j] = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} x[i-u, j-v] \cdot w[u,v]$
    - Relaxed definition

      $z[i,j] = \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} x[i+u, j+v] \cdot w[u,v]$

Lecture 10/11 Ensemble Methods

- Recall decision trees
  - + Computationally efficient
  - + Interpretable
  - + Compatible with continuous, discrete features
  - - Prone to overfitting, addressible by heuristics (e.g., pruning)
  - - High bias, addressible by bagging
  - - High variance (especially short trees), addressible by boosting

- Bias-Variance Tradeoff

True error of model = Bias + Variance

- Bias:
  - How well can model approximate target
  - Decrease with model complexity
- Variance:
  - How well can model approximate anything
  - Increase with model complexity

- Bagging
  - Bootstrap aggregating
  - Combine prediction of many hypotheses to reduce variance
    - Regression $\bar{h}(\vec{x}) = \frac{1}{m}\sum_{i=1}^{m} h_i(\vec{x})$
    - Classification $\bar{h}(\vec{x}) = \text{sign}(\frac{1}{m}\sum_{i=1}^{m} h_i(\vec{x}))$ where encoding is $\{+1, -1\}$
  - If $x_1, \dots, x_n$ independent rvs with var σ^2 then var of $\frac{1}{n}\sum_{i=1}^{n} x_i$ is $\frac{\sigma^2}{n}$

- Random Forests
  - Input: $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\}$, B
  - Alg:

```
for b=1 to B:
    D_b ← sample n points from D with replacement
    t_b ← learn decision tree using D_b, ID3 algorithm with split feature randomization
```

  - Output: $\bar{t}$ the aggregated hypothesis
  - Important because bagging alone does not create independent trees

- Boosting
  - Tries to reduce bias of a "weak" or highly biased model
  - Can also reduce variance

- AdaBoost [Schapire 1989]
  - Intuition: iteratively reweight inputs, giving more weight to inputs that are difficult to predict correctly
  - Most widely known/used in practice today

Lecture 10/11 cont.

- ## AdaBoost
  - Input: $D(Y=\{-1,+1\}), T$
  - Algorithm:

```
    $w_1^{(0)}, \dots, w_n^{(0)} \leftarrow \frac{1}{n}$
    for $t=1$ to $T$:
        $h_t \leftarrow$ train weak learner minimizing weighted training error $\varepsilon_t$
weighted training error of $h_t$    $\varepsilon_t \leftarrow \sum_{i=1}^n w_i^{(t-1)} \mathbb{1}(h_t(\vec{x}_i) \neq y_i)$
importance of $h_t$    $\alpha_t \leftarrow \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
weights update    $w_i^{(t)} \leftarrow \frac{w_i^{(t-1)}}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(\vec{x}_i)=y_i \\ e^{\alpha_t} & \text{if } h_t(\vec{x}_i)\neq y_i \end{cases} = \frac{1}{Z_t} w_i^{(t-1)} \exp(-\alpha_t y_i h_t(\vec{x}_i))$
```

  - Output: aggregated hypothesis
    - $g_T(\vec{x}) = \text{sign}(H_T(\vec{x})) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(\vec{x})\right)$
  - α_t intuition: want good weak learners to have large weights
  - w_i intuition: want incorrectly classified inputs to receive a higher weight in the next round
    - $\varepsilon_t < \frac{1}{2} \Rightarrow \frac{1-\varepsilon_t}{\varepsilon_t} > 1 \Rightarrow \alpha_t > 0 \Rightarrow e^{-\alpha_t} < 1$ and $e^{\alpha_t} > 1$
  - AdaBoost intuition:
    - Want weak learners because low variance/cheap to compute
    - Want final hypothesis to be weighted combination of weak learners because individual weak learners do poorly
    - AdaBoost greedily minimizes exponential loss $e(h, \vec{x}, y) = e^{-yh(\vec{x})}$ which upper bounds the binary error

- ## Exponential loss
  - Claim: $\frac{1}{n}\sum_{i=1}^n e^{-y_i h(\vec{x}_i)} \geq \frac{1}{n}\sum_{i=1}^n \text{sign}(h(\vec{x}_i) \neq y_i)$
  - Then: $\frac{1}{n}\sum_{i=1}^n e^{-y_i h(\vec{x}_i)} \rightarrow 0 \geq \frac{1}{n}\sum_{i=1}^n \text{sign}(h(\vec{x}_i) \neq y_i) \rightarrow 0$
  - Claim: If $g_T = \text{sign}(H_T)$ is AdaBoost hypothesis then $\frac{1}{n}\sum_{i=1}^n e^{-y_i H_T(\vec{x}_i)} = \prod_{t=1}^T Z_t$
    - Consider $w_i^{(0)}, w_i^{(1)}, w_i^{(2)}$.
    - In general, $w_i^{(T)} = \frac{\prod_{t=1}^T \exp(-\alpha_t y_i h_t(\vec{x}_i))}{n \prod_{t=1}^T Z_t}$

$$= \frac{\exp[-(y_i \sum_{t=1}^T \alpha_t h_t(\vec{x}_i))]}{n \prod_{t=1}^T Z_t}$$

$$= \frac{\exp(-y_i H_T(\vec{x}_i))}{n \prod_{t=1}^T Z_t}$$

    - Since normalized, $\sum_{i=1}^N w_i^{(T)} = 1 = \sum_{i=1}^N \frac{\exp(-y_i H_T(\vec{x}_i))}{n \prod_{t=1}^T Z_t}$

      Hence $\prod_{t=1}^T Z_t = \frac{1}{n}\sum_{i=1}^N \exp(-y_i H_T(\vec{x}_i))$
  - Then: one way of minimizing in-sample exponential loss is to greedily minimize Z_t

- ## Greedy exponential loss minimization

$$Z_t = \sum_{i=1}^N w_i^{(t-1)} e^{-w y_i h(\vec{x})}$$

$$= \sum_{y_i = h_t(\vec{x}_i)} w_i^{(t-1)} e^{-w} + \sum_{y_i \neq h_t(\vec{x}_i)} w_i^{(t-1)} e^{w}$$

$$= e^{-w}(1-\varepsilon_t) + e^{w}(\varepsilon_t)$$

$$\frac{\partial Z_t}{\partial w} = -e^{-w}(1-\varepsilon_t) + e^{w}(\varepsilon_t)$$

$= 0$ when $w = w^*$ for some w^*

$$e^{w^*}(\varepsilon_t) = e^{-w^*}(1-\varepsilon_t)$$

$$e^{2w^*} = \frac{1-\varepsilon_t}{\varepsilon_t}$$

$$w^* = \frac{1}{2}\log\frac{1-\varepsilon_t}{\varepsilon_t}$$

i.e., importance defined to minimize Z_t, minimizes since $\frac{\partial^2 Z_t}{\partial w^2} > 0$

Z_t simplifies further to $2\sqrt{\varepsilon_t(1-\varepsilon_t)} < 1$ if $\varepsilon_t < \frac{1}{2}$

Lecture 10/11 cont.

- Training error

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(y_i \neq g_T(\vec{x}_i)) \leq \prod_{t=1}^{T} Z_t$$
$$= \prod_{t=1}^{T} 2\sqrt{\varepsilon_t(1-\varepsilon_t)} \to 0 \text{ as } T\to\infty \text{ as long as } \varepsilon_t < \tfrac{1}{2}\ \forall t$$

- True error [Freund, Schapire, 1995]
  - For AdaBoost,

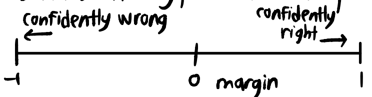
$$\text{True Error} \leq \text{Training Error} + \tilde{O}\left(\sqrt{\frac{d_{VC}(H)\ T}{n}}\right)$$

where $d_{VC}(H)$ = VC dimension of weak learners, T = # weak learners, n = # training data points
  - Empirically, increasing T does not lead to overfitting as implied by above bound

- Margin of training point $(\vec{x}_i, y_i)$ defined as

$$m(\vec{x}_i, y_i) = \frac{y_i \sum_{t=1}^{T} \alpha_t h_t(\vec{x}_i)}{\sum_{t=1}^{T} \alpha_t}$$

  - How confident g_T is in its prediction



  - Schapire's observation
    - Boosting increases margin even after training error reaches 0

Lecture 10/13 Clustering & k-Means

- Learning Paradigms

Supervised	$D = \{\vec{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ $\quad \vec{x} \sim p^*(\cdot)$ $\quad y \sim c^*(\cdot)$
Regression	$y^{(i)} \in \mathbb{R}$
Classification	$y^{(i)} \in \{1, \dots, K\}$
Binary classification	$y^{(i)} \in \{-1, +1\}$
Structured Prediction	$\vec{y}^{(i)}$ a vector
Unsupervised	$D = \{\vec{x}^{(i)}\}_{i=1}^{N}$ $\quad \vec{x} \sim p^*(\cdot)$
Semi-supervised	$D = \{\vec{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\vec{x}^{(j)}\}_{j=1}^{N_2}$
Online	$D = \{(\vec{x}^{(1)}, y^{(1)}), (\vec{x}^{(2)}, y^{(2)}), \dots\}$
Active Learning	$D = \{\vec{x}^{(i)}\}_{i=1}^{N}$, can query $y^{(i)} = c^*(\cdot)$ at a cost
Imitation Learning	$D = \{(s^{(1)}, a^{(1)}), (s^{(2)}, a^{(2)}), \dots\}$
Reinforcement Learning	$D = \{(s^{(1)}, a^{(1)}, r^{(1)}), (s^{(2)}, a^{(2)}, r^{(2)}), \dots\}$

- Clustering
  - Automatically partition unlabeled data into groups of similar datapoints

- Distance measures

$D(A,B) = D(B,A)$	symmetry
$D(A,A) = 0$	constancy of self-similarity
$D(A,B) = 0$ iff $A = B$	positivity separation
$D(A,B) \leq D(A,C) + D(C,B)$	triangle inequality

Lecture 10/13 cont.

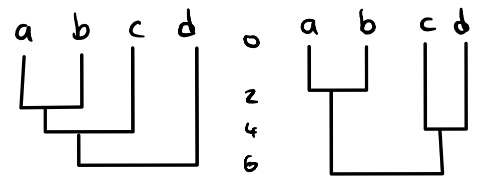
- Minkowski metric
  - Given $\vec{x} = (x_1, \dots, x_p)$, $\vec{y} = (y_1, \dots, y_p)$
  - Minkowski metric is $d(\vec{x}, \vec{y}) = (\sum_{i=1}^{p} |x_i - y_i|^r)^{1/r}$
    - $r=2$, Euclidean distance
    - $r=1$, Manhattan distance
    - $r=+\infty$, "sup" distance $\max_{1 \le i \le p} |x_i - y_i|$

- Hamming distance
  - Manhattan distance when all features binary

- Edit distance

- Clustering algorithms
  - Hierarchical
    - Bottom up : agglomerative
    - Top down : divisive
  - Partition
    - k-means clustering
    - Mixture-Model based clustering

- Agglomerative clustering
  ① Each object in a separate cluster
  ② Repeat:
    ⓐ Join most similar pair of clusters
    ⓑ Update similarity of new cluster to others until there is only one cluster
  - Greedy alg, less accurate but simple implementation
  - Single-Linkage = nearest neighbor, similarity on closest members → allow anisotropic, non-convex shapes
  - Complete-Linkage = furthest neighbor, similarity on furthest members → assume isotopic, convex shapes
  - Centroid = similarity between center of gravity
  - Average-Linkage = average similarity of all cross-cluster pairs
  - Dendrogram, convenient way to visualize clusters



- Divisive clustering
  ① All data in single cluster
  ② Repeat until each object a separate cluster
    ⓐ Split each cluster into two using partition algorithm
  - More accurate, more complex implementation

Lecture 10/13 cont.

- Partitioning Algorithms
  - Input: set of objects, number K
  - Goal: partition of K clusters optimizing chosen partitioning criterion
    - Global optimal: exhaustive enumeration of partitions
    - Effective heuristic: k-means

- k-means
  - Input: $D = \{x^{(i)}\}_{i=1}^{N}$, K
  - Algorithm

```
    Initialize K cluster centers (e.g., randomly)
    Iterate
        Assign points to nearest cluster centers
        Reestimate K cluster centers (aka centroid or mean) by assuming above assignments correct
    Until
        No objects changed membership in last iteration
```

- Optimizing k-means
  - Input: $\vec{x}^{(1)}, \dots, \vec{x}^{(N)}$, K, $\vec{x}^{(i)} \in \mathbb{R}^m$
  - Output: $z^{(1)}, \dots, z^{(N)}$, $z^{(i)} \in \{1, \dots, k\}$ "cluster assignments per point"
  - Output: $\mu_1, \dots, \mu_K$, $\mu_i \in \mathbb{R}^m$

$$\hat{\mu}_1, \dots, \hat{\mu}_K = \text{argmin}_{\mu_1, \dots, \mu_K} \sum_{i=1}^{N} \min_{j \in \{1 \dots K\}} \|x^{(i)} - \mu_j\|_2^2$$
$$= \text{argmin}_{\mu_1, \dots, \mu_K, z^{(1)}, \dots, z^{(N)}} \sum_{i=1}^{N} \|x^{(i)} - \mu_{z^{(i)}}\|_2^2$$

  - Computational complexity
    - If $N \geq 2$, $K \geq 2$, not convex, NP hard

- Alternating minimization

ⓐ $z = \text{argmin}_z \sum_{i=1}^{N} \|x^{(i)} - \mu_{z^{(i)}}\|_2^2$

ⓑ $\mu_1, \dots, \mu_K = \text{argmin}_{\mu_1, \dots, \mu_K} \sum_{i=1}^{N} \|x^{(i)} - \mu_{z^{(i)}}\|_2^2$

ⓐ $z^{(1)} = \text{argmin}_{z^{(1)} \in \{1, \dots, K\}} \|x^{(i)} - \mu_{z^{(i)}}\|_2^2$ ⓑ $\mu_1 = \text{argmin}_{\mu_1} \sum_{i: z^{(i)}=1} \|x^{(i)} - \mu_{z^{(i)}}\|_2^2$

$\vdots$ $\vdots$

- Coordinate descent
  - Two approaches to $\min_{\theta_1, \theta_2} J(\theta_1, \theta_2)$
  - Step based on derivative for one parameter

$\theta_1 \leftarrow \theta_1 - \eta \, \partial J / \partial \theta_1$

$\theta_2 \leftarrow \theta_2 - \eta \, \partial J / \partial \theta_2$

  - Find minimum for one parameter

$\theta_1 \leftarrow \text{argmin}_{\theta_1} J(\theta_1, \theta_2)$

$\theta_2 \leftarrow \text{argmin}_{\theta_2} J(\theta_1, \theta_2)$

- Block coordinate descent
  - $\vec{\alpha}, \vec{\beta}$ instead of θ_1, θ_2
  - Like above

Lecture 10/13 cont.

- Computational complexity
  - Each iteration
    - Computing cluster centers: each object added once to cluster center $O(N)$
    - Computing $\forall$obj $\forall$cluster center dist(obj, cluster center), $O(KN)$
  - ℓ iterations $\Rightarrow O(\ell KN)$

- Seed choice matters
  - k-means always converges but may converge to local optimum, arbitrarily worse

- K-mediod (median)
  - Use median training point as cluster center
  - More robust to outliers pulling mean away
  - Better interpretability
  - More work to compute

- Choosing K
  - Look for knee in objective function

- More k-means issues
  - Clusters may overlap
  - Clusters may be "wider" than others
  - Clusters may not be linearly separable

(next page)

Lecture 10/18 Gaussian Mixture Models & Expectation-Maximization

Categorical Distribution

- When discrete rv represents multiple possible classes and the probability of each class occurring
- $Y = \{1, \dots, K\}$, but $\mathbb{E}[Y]$ does not make sense
- $Y = [Y_1, \dots, Y_K]^T$ where each Y_k binary, Y one-hot
  - $Y \sim \text{Categorical}(\pi_1, \dots, \pi_K)$ where $0 \leq \pi_i \leq 1$, $\sum_{i=1}^{K} \pi_i = 1$
  - $\mathbb{E}[Y] = [\mathbb{E}[Y_1], \dots, \mathbb{E}[Y_K]]^T = [\pi_1, \dots, \pi_K] = \vec{\pi}$

Categorical Gaussian Generative Model

- Estimating parameters
  - $Y \sim \text{Categorical}(\pi_1, \pi_2, \pi_3)$
  - $X_{Y=k} \sim N(\mu_k, \sigma_k^2)$
  - $D = \{x^{(i)}, y^{(i)}\}_{i=1}^{N}$
  - $$\begin{aligned}\hat{\theta}_{MLE} &= \underset{\theta}{\text{argmax}} \sum_{i=1}^{N} \log P(x^{(i)}, y^{(i)} \mid \theta) \\ &= \underset{\theta}{\text{argmax}} \sum_{i=1}^{N} \log P(y^{(i)} \mid \theta) P(x^{(i)} \mid y^{(i)} \mid \theta) \\ &= \underset{\theta}{\text{argmax}} \sum_{i=1}^{N} \log \prod_{k=1}^{K} \pi_k^{y_k^{(i)}} f_N(x^{(i)} \mid \mu_k, \sigma_k^2)^{y_k^{(i)}} \\ &= \underset{\theta}{\text{argmax}} \sum_{i=1}^{N} \sum_{k=1}^{K} y_k^{(i)} \log \pi_k f_N(x^{(i)} \mid \mu_k, \sigma_k^2)\end{aligned}$$
- Inference
  - $$\begin{aligned}P(y_k = 1 \mid x, \theta) &= \frac{P(x, y_k = 1 \mid \theta)}{P(x \mid \theta)} \\ &= \frac{P(x, y_k = 1 \mid \theta)}{\sum_{j=1}^{K} P(x, y_j = 1 \mid \theta)} \\ &= \frac{P(y_k = 1 \mid \theta) P(x \mid y_k = 1)}{\sum_{j=1}^{K} P(x, y_j = 1 \mid \theta)} \\ &= \frac{\pi_k f_N(x \mid \mu_k, \sigma_k^2)}{\sum_{j=1}^{K} \pi_j f_N(x \mid \mu_j, \sigma_j^2)}\end{aligned}$$

k-means bad case example

- Overlapping clusters
- Some clusters can be "wider"
- Clusters may not be linearly separable

Partitioning Algorithms

- k-means is a **hard assignment**: each object belongs to only one cluster
- **Mixture modeling** is a **soft assignment**: probability that an object belongs to a cluster

Gaussian Mixture Model

- Mixture of **K Gaussian distributions** (multimodal distribution)

  $p(x \mid z_k = 1) \sim N(\mu_k, \Sigma_k)$

  $$p(x) = \sum_{k=1}^{K} \underbrace{p(x \mid z_k = 1)}_{\text{mixture component}} \underbrace{P(z_k = 1)}_{\text{mixture proportion}}$$

- K components, component k generates data from Gaussian(μ_k, Σ_k)
- Each data point generated as follows:
  1. Pick component k with probability $p(z_k = 1)$
  2. Data point $x \sim N(\mu_k, \Sigma_k)$

Lecture 10/18 cont.

- Learning general Gaussian Mixture Models
  - GMM
    - $x_1, \dots, x_M \sim p(x) = \sum_{k=1}^{K} p(x \mid z_k = 1)\, p(z_k = 1)$
    - Mixture $\pi_k = p(z_k = 1)$
    - Gaussian components $p(x \mid z_k = 1) \sim N(\mu_k, \Sigma_k)$
    - Parameters $\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$
  - Estimating parameters
    - MLE?
      - $\underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{N} p(x^{(i)} \mid \theta) = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{N} \sum_{k=1}^{K} \pi_k \lvert \Sigma_k \rvert^{-\frac{1}{2}} e^{-\frac{1}{2}(x^{(i)}-\mu_k)^T \Sigma_k (x^{(i)}-\mu_k)}$
      - $\frac{\partial}{\partial \mu_k} \ell(\theta; D) = 0$, etc $\Rightarrow$ no closed form
      - Gradient descent $\Rightarrow$ possible but complicated, often slow, need to consider constraints on parameters
        - $\frac{\partial \ell}{\partial \theta} = \sum_{k=1}^{K} r_k \frac{\partial \ell_k}{\partial \theta_k}$ the responsibility weighted sum of individual log likelihood gradients
        - For constraints, use constrained optimization or reparameterize (e.g., softmax, Cholesky decomposition $\Sigma^{-1} = A^T A$)

Lecture 10/20 GMM, EM, PCA

- Recall Log Likelihood vs Complete Log Likelihood
  - LL : $D = \{\vec{x}^{(i)}\}_{i=1}^{N}$
    - $\ell(\theta \mid D) = \log \prod_{i=1}^{N} p(\vec{x}^{(i)} \mid \theta)$
    - $= \sum_{i=1}^{N} \log \sum_{k=1}^{K} p(\vec{x}^{(i)}, z^{(i)} \mid \theta)$
    - stuck
  - CLL: $D = \{\vec{x}^{(i)}, \vec{z}^{(i)}\}_{i=1}^{N}$
    - $\ell_c(\theta \mid D_c) = \log \prod_{i=1}^{N} p(\vec{x}^{(i)}, \vec{z}^{(i)} \mid \theta)$
    - $= \sum_{i=1}^{N} \log \prod_{k=1}^{K} \pi_k^{z_k^{(i)}} f_N(x^{(i)} \mid \mu_k, \Sigma_k)^{z_k^{(i)}}$ indicator trick
    - $= \sum_{i=1}^{N} \sum_{k=1}^{K} z_k^{(i)} \log(\pi_k f_N(x^{(i)} \mid \mu_k, \Sigma_k))$

- Expected value of Complete Log Likelihood
  - Replace z with $\mathbb{E}_{z \mid x, \theta}[\ell_c(\theta \mid D_c)]$
    - $= \mathbb{E}_{z \mid x, \theta}[\sum_{i=1}^{N} \sum_{k=1}^{K} z_k^{(i)} \log(\pi_k f_N(x \mid \mu, \theta))$
    - $= \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbb{E}_{z \mid x, \theta}[z_k^{(i)}] \log(\pi_k f_N(x \mid \mu, \theta))$, note $\mathbb{E}_{z \mid x, \theta}[z_k^{(i)}] = \sum_{z_k \in \{0,1\}} z_k\, p(z_k \mid x^{(i)}, \theta) = p(z_k = 1 \mid x^{(i)}, \theta)$
    - $= \sum_{i=1}^{N} \sum_{k=1}^{K} p(z_k^{(i)} = 1 \mid x^{(i)}, \theta) \log(\pi_k f_N(x \mid \mu, \theta))$

- Expectation-Maximization (EM) for GMMs
  - Initialize
    - $t = 0, \pi_k^{(0)}, \vec{\mu}_k^{(0)}, \Sigma_k^{(0)}$
  - E-step
    - For fixed GMM parameters $\theta^{(t)}$, update probability point $\vec{x}^{(i)}$ belongs to cluster k, $p(z_k^{(i)} = 1 \mid \vec{x}^{(i)}, \theta^{(t)})$
  - M-step
    - For fixed $p(z_k^{(i)} = 1 \mid \vec{x}^{(i)}, \theta^{(t)})$, update estimates for $\pi_k^{(t+1)}, \vec{\mu}_k^{(t+1)}, \Sigma_k^{(t+1)}$
  - Iterate between E and M steps

- EM notes
  - EM = optimization strategy for objective functions that can be interpreted as likelihoods in the presence of missing data
  - Simpler than gradient methods – no step size, enforces constraints, call inference and fully observed learning as subroutines

Lecture 10/20 cont.

- More EM notes
  - EM iterative
    - E-step: fill in hidden values using inference $p(z|x,\theta)$
    - M-step: update parameters $t+1$ using standard MLE/MAP method applied to completed data
  - Monotonically improves or stays unchanged
    - Will always converge to local optimum of the likelihood

- EM for GMM
  - E
    $\mathbb{E}_{z|x,\theta^{(t)}}[z_k^{(i)}] = p(z_k^{(i)}=1 \mid x^{(i)}, \theta^{(t)})$
  - M
    $$\left.\begin{matrix}\pi_k^{(t+1)} \\ \mu_k^{(t+1)} \\ \Sigma_k^{(t+1)}\end{matrix}\right\} \operatorname{argmax}_\theta \mathbb{E}_{z|x,\theta^{(t)}}\left[\ell_c(\theta|D_c)\right]$$
  - Specifically

    E: $$p(z_k^{(i)}=1 \mid x^{(i)}, \theta^{(t)}) \leftarrow \frac{\pi_k^{(t)} N(x^{(i)}; \mu_k^{(t)}, \Sigma_k^{(t)})}{\sum_{j=1}^K \pi_j^{(t)} N(x^{(i)}; \mu_j^{(t)}, \Sigma_j^{(t)})} \quad \forall i,k$$

    M: $$\pi_k^{(t+1)} \leftarrow \frac{\sum_{i=1}^N P(z_k^{(i)}=1 \mid x^{(i)}, \theta^{(t)})}{N} \quad \forall k$$

    $$\mu_k^{(t+1)} \leftarrow \frac{\sum_{i=1}^N P(z_k^{(i)}=1 \mid x^{(i)}, \theta^{(t)})\, x^{(i)}}{\sum_{i=1}^N P(z_k^{(i)}=1 \mid x^{(i)}, \theta^{(t)})} \quad \forall k$$

    $$\Sigma_k^{(t+1)} \leftarrow \frac{\sum_{i=1}^N P(z_k^{(i)}=1 \mid x^{(i)}, \theta^{(t)})\,(x^{(i)}-\mu_k^{(t+1)})(x^{(i)}-\mu_k^{(t+1)})^T}{\sum_{i=1}^N P(z_k^{(i)}=1 \mid x^{(i)}, \theta^{(t)})} \quad \forall k$$

- General EM theory
  - Recall MLE, learn θ maximizing $\ell_c(\theta; D)$
  - But z not observed, computing $\ell(\theta; D) = \log \sum_z p(x,z|\theta) = \log \sum_z p(z|\theta_z)\, p(x|z,\theta_x)$ hard
  - If z observable, then $\ell_c(\theta; x,z) = \log p(x,z|\theta)$, solvable with standard MLE
  - But z isn't. So $\ell(\theta; x) = \log \sum_z p(x,z|\theta)$.

- Expected complete log likelihood
  - For any distribution $q(z)$, define expected complete log likelihood
    $\langle \ell_c(\theta; x,z)\rangle_q = \sum_z q(z|x,\theta) \log p(x,z|\theta)$
  - Deterministic function of θ
  - Linear in $\ell_c(\cdot)$
  - By Jensen's inequality
    $$\begin{aligned}\ell(\theta;x) &= \log p(x|\theta) \\ &= \log \sum_z p(x,z|\theta) \\ &= \log \sum_z q(z|x) \frac{p(x,z|\theta)}{q(z|x)} \\ &\geq \sum_z q(z|x) \log \frac{p(x,z|\theta)}{q(z|x)}\end{aligned}$$
    i.e., $\ell(\theta;x) \geq \langle \ell_c(\theta;x,z)\rangle_q + H_q$

- Free energy
  - For fixed data x, define functional $F(q,\theta) = \sum_z q(z|x) \log \frac{p(x,z|\theta)}{q(z|x)} \leq \ell(\theta;x)$
  - EM is coordinate ascent on F, E: $q^{t+1} = \operatorname{argmax}_q F(q,\theta^t)$, M: $\theta^{t+1} = \operatorname{argmax}_\theta F(q^{t+1}, \theta^t)$

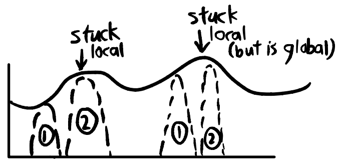
Lecture 10/20 cont.

- E maximizes expected ℓ_c wrt q
  - Claim: $q^{t+1} = \text{argmax}_q F(q, \theta^t) = p(z|x,\theta^t)$
  - Proof
    $$F(p(z|x,\theta^t),\theta^t) = \Sigma_z p(z|x,\theta^t) \log \frac{p(x,z|\theta^t)}{p(z|x,\theta^t)}$$
    $$= \Sigma_z p(z|x,\theta^t) \log p(x|\theta^t)$$
    $$= \log p(x|\theta^t) = \ell(\theta^t; x)$$
    Hence $\ell(\theta^t;x) \geq F(q,\theta)$
  - Or prove by variational calculus
  - Or prove by $\ell(\theta;x) - F(q,\theta) = KL(q \| p(z|x,\theta))$

- E plugs in the posterior expectation of latent variables
  - WLOG assume $p(x|z,\theta)$ generalized exponential family distribution
    $p(x|z,\theta) = \frac{1}{z(\theta)} h(x,z) \exp\{\Sigma_i \theta_i f_i(x,z)\}$
    - If $p(X|Z)$ GLIM then $f_i(x,z) = \eta_i^T(z) \xi_i(x)$
  - Then expected complete log likelihood
    $$\langle \ell_c(\theta^t; x,z)\rangle_{q^{t+1}} = \Sigma_z q(z|x,\theta^t) \log p(x,z|\theta^t) - A(\theta)$$
    $$= \Sigma_i \theta_i^t \langle f_i(x,z)\rangle_{q(z|x,\theta^t)} - A(\theta)$$
    $$\overset{p\sim GLIM}{=} \Sigma_i \theta_i^t \langle \eta_i(z)\rangle_{q(z|x,\theta^t)} \xi_i(x) - A(\theta)$$

- M maximizes expected ℓ_c wrt θ
  - Notice H_q term does not depend on θ
  - $\theta^{t+1} = \text{argmax}_\theta \langle \ell_c(\theta; x,z)\rangle_{q^{t+1}} = \text{argmax}_\theta \Sigma_z q(z|x) \log p(x,z|\theta)$

- EM: use multiple randomized initializations in practice



Dimensionality reduction
- Find $f: \mathbb{R}^M \to \mathbb{R}^K$, $g: \mathbb{R}^K \to \mathbb{R}^M$, $K << M$, $f(\vec{x}) = \vec{z}$, $\vec{x}' = g(\vec{z})$
  minimizing reconstruction error, i.e., $\min_{f,g} \Sigma_{i=1}^N \| x^{(i)} - x'^{(i)} \|_2^2$

Principal Component Analysis (PCA)
- If data on/near low-dimensional subspace, then axes of this subspace are effective representation of the data
- Input data $D = \{x^{(i)}\}_{i=1}^N$, $X = \begin{bmatrix} x^{(1)T} \\ \vdots \\ x^{(N)T} \end{bmatrix}$
  - Assume data centered, i.e., $\mu = \frac{1}{N}\Sigma_{i=1}^N x^{(i)} = 0$
  - Otherwise subtract sample mean to center
- Sample covariance matrix
  - $\Sigma_{jk}\ \frac{1}{N}\Sigma_{i=1}^N (x_j^{(i)} - \mu_k)(x_k^{(i)} - \mu_j)$
  - Since centered, $\Sigma = \frac{1}{N} X^T X$
- PCA Alg, input X, X_{test}, K
  1. Center data and scale each axis based on training data $\to X, X_{test}$
  2. $V = \text{eigenvectors}(X^T X)$
  3. Keep only top eigenvectors V_K
  4. $Z_{test} = X_{test} V_K$
  5. Optionally V_K^T rotates Z_{test} back to subspace X'_{test}, uncenter

Lecture 10/20 cont.

- Principal components
  - The i^{th} principal component is eigenvector of X^TX associated with i^{th} largest eigenvalue λ_i
  - Recall $\lambda_1 \geq \lambda_2 \geq \dots$
  - M dimensions $\Rightarrow X^TX_{M\times M} \Rightarrow$ up to M eigenvectors $\Rightarrow$ M principal components

Rotation
- For any orthogonal $V \in \mathbb{R}^{M\times M}$
- Rotate $z^{(i)} = Vx^{(i)}$
- Unrotate $x'^{(i)} = V^T z^{(i)}$

Projection
- Reconstruction error
  - $\|x^{(i)} - x'^{(i)}\|_2^2$
  - $v^* = \text{argmin}_{v,\ st\ \|v\|_2=1} \sum_{i=1}^N \|x^{(i)} - (v^Tx^{(i)})v\|_2^2$
- Variance of projection
  - $v^* = \text{argmax}_{v,\ st\ \|v\|_2=1} \sum_{i=1}^N v^T x'^{(i)}$

Lecture 10/25 PCA, Kernel PCA, Autoencoders, Independent Component Analysis

- Midterm up to not including kernel PCA

Maximizing Variance = Minimizing Reconstruction Error
- $\|\vec{x}^{(i)} - (\vec{v}^Tx^{(i)})\vec{v}\|_2^2 = \|\vec{x}^{(i)}\|^2 - (\vec{v}^T\vec{x}^{(i)})^2$ since $\vec{v}^T\vec{v} = \|\vec{v}\|_2^2 = 1$
- $\vec{v}^* = \text{argmin}_{v:\|v\|^2=1} \frac{1}{N}\sum_{i=1}^N \|\vec{x}^{(i)} - (\vec{v}^T\vec{x}^{(i)})\vec{v}\|_2^2$
  $= \text{argmin}_{v:\|v\|^2=1} \frac{1}{N}\sum_{i=1}^N \|\vec{x}^{(i)}\|^2 - (\vec{v}^T\vec{x}^{(i)})^2$
  $= \text{argmax}_{v:\|v\|^2=1} \frac{1}{N}\sum_{i=1}^N (\vec{v}^T\vec{x}^{(i)})^2$

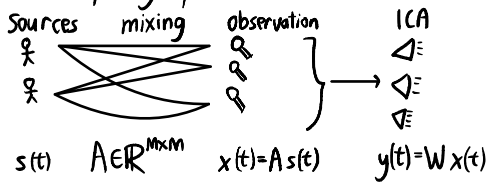
Lagrange multipliers for PCA
- Σ symmetric
- $\hat{v} = \text{argmax}_{\vec{v}}\ \vec{v}^T\Sigma\vec{v}$
  such that $\|v\|_2^2 = 1$
- $L(\vec{v}, \lambda) = \vec{v}^T\Sigma\vec{v} - \lambda(\vec{v}^T\vec{v} - 1)$
- $\nabla_{\vec{v}} L(\vec{v}, \lambda) = \vec{v}^T(\Sigma + \Sigma^T) - 2\lambda\vec{v}^T$
  $= 2\vec{v}^T\Sigma - 2\vec{v}^T\lambda$
- $\nabla_{\vec{v}} L(\vec{v},\lambda) = 0$ when $\lambda = \Sigma$
- So Σ eigenvalues, $\vec{v}$ eigenvectors

SVD for PCA
- $X = USV^T$, $A \in \mathbb{R}^{N\times M}$
- $U_{N\times N}$ orthogonal, cols = left singular vectors of A, cols = eigenvectors XX^T
- $V_{M\times M}$ orthogonal, cols = right singular vectors of A, cols = eigenvectors X^TX
- $S_{N\times M}$ diagonal, diagonals = singular entries of X, σ_k. Each σ_k^2 are eigenvalues for XX^T and X^TX

Lecture 10/25 cont.

- Kernel maps
  - Feature maps + kernel trick → linearize data for PCA
  - Input: X, X_{test}
  - Algorithm
    1. Compute kernel matrix $K_{ij} = k(X_i, X_j) = \phi(X_i)^T \phi(X_j)$
    2. "Center K"
    3. U = eigenvectors(K), λ = eigenvalues(K)
    4. $\alpha_j = \frac{1}{\sqrt{\lambda_j}} u_j$
    5. $z_{test,j} = \sum_i \alpha_{ij}\, k(X_i, X_{test})$

- Independent Component Analysis (ICA)
  - Cocktail party problem



- ICA vs PCA

PCA	ICA
$X = US$, $U^TU = I$	$X = AS$, $\exists A^{-1}$
Compression ✓, $M<N$	Compression ✗, $M=N$
Remove correlation ✓	Remove correlation ✓
Remove higher order dependence ✗	Remove higher order dependence ✓
Components: bigger eigenvalue, more importance	Components equally important

- ICA
  - Given x
    - Find $y = \hat{s}$, $W = \hat{A}^{-1}$
  - Solution $y = Wx$
    - Remove mean so $E[X] = 0$
    - Whitening $E[xx^T] = I$
    - Find orthogonal W optimizing objective

- See slides for Kurtosis maximization, FastICA, whitening visualization

---

Lecture 11/01 Computational Learning Theory

- Exam
  - ~70 mean
  - Usually 50% A

Lecture 11/01 cont.

- Zero train error ⇒ realizable case for true error
- Low train error ⇒ agnostic case for true error
- Theoretical justification for regularization

- Risk = expected loss over data points
- 0-1 loss = cost of one when misclassifying a point

- True error (expected risk)
  - $R(h) = P_{x \sim p^*(x)}(c^*(x) \neq h(x))$ c is oracle
- Train error (empirical risk)
  - $\hat{R}(h) = P_{x \sim S}(c^*(x) \neq h(x))$ $x \sim S$ means x sampled from empirical distribution
    $= \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(c^*(x^{(i)}) \neq h(x^{(i)}))$
    $= \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(y^{(i)} \neq h(x^{(i)}))$

- PAC/SLT model
  - Probably Approximately Correct / Statistical Learning Theory
    1. Generate $\vec{x}^{(i)} \sim p^*(\vec{x})$ $\forall i$ p^* is an unknown distribution
    2. Oracle $y^{(i)} = c^*(\vec{x}^{(i)})$ $\forall i$
    3. Learning alg picks $\hat{h} = \text{argmin}_{h \in H} \hat{R}(h)$
  - Goal: choose h with low generalization error $R(h)$

- Hypothesis functions
  - True function c^*
  - Expected risk minimizer $h^* = \text{argmin}_{h \in H} R(h)$
  - Empirical risk minimizer $\hat{h} = \text{argmin}_{h \in H} \hat{R}(h)$

- Bounding $R(h)$ in terms of $\hat{R}(h)$
  - PAC learner yields $h \in H$, $R(h) \approx 0$ (approx correct), with high probability $P(R(h) \approx 0) \approx 1$
  - PAC criterion
    $P(|R(h) - \hat{R}(h)| \leq \varepsilon) \geq 1 - \delta$
  - PAC learner consistent if
    - $\forall \varepsilon, \forall \delta$
    - $\exists N$ such that for any p^*
    - $P(|R(h) - \hat{R}(h)| > \varepsilon) < \delta$
  - N above is the sample complexity
    - N finite ⇒ H learnable
    - N is poly in $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$ ⇒ H PAC learnable

- Sample complexity

	Realizable	Agnostic
Finite $\lvert H \rvert$	$N \geq \frac{1}{\varepsilon}\left(\log \lvert H \rvert + \log \frac{1}{\delta}\right)$	$N \geq \frac{1}{2\varepsilon^2}\left(\log \lvert H \rvert + \log \frac{2}{\delta}\right)$
Infinite $\lvert H \rvert$	$N = O\left(\frac{1}{\varepsilon}\left[VC(H) \log \frac{1}{\varepsilon} + \log \frac{1}{\delta}\right]\right)$	$N = O\left(\frac{1}{\varepsilon^2}\left(VC(H) + \log \frac{1}{\delta}\right)\right)$

So that with probability $1-\delta$ for all $h \in H$ (with $\hat{R}(h) = 0$ we have $R(h) \leq \varepsilon$) ← finite $|H|$
(we have $|R(h) - \hat{R}(h)| \leq \varepsilon$) ← infinite $|H|$

Lecture 11/01 cont.

- Proof for finite $|H|$, realizable
  - $h \in H$ consistent if $R(h)=0$
  - Assume k bad hypotheses $h_1 \cdots h_k$ with $R(h_i) > \varepsilon$
  - Pick bad h_i, $P(h_i$ consistent with first training point$) \leq 1-\varepsilon$
    - $P(h_i$ consistent with first N training points$) \leq (1-\varepsilon)^N$
    - $= $ "$\hat{R}(h_i)=0$"
  - $P($at least one h_i consistent with first N training points$) \leq k(1-\varepsilon)^N$ by union bound
    - $\leq |H|(1-\varepsilon)^N$, recall $1-x \leq e^{-x}$
    - $\leq |H| e^{-\varepsilon N}$
  - Fix δ, calculate N such that $|H| e^{-\varepsilon N} \leq \delta$

Lecture 11/03 More Learning Theory

- Continue proof

$$P(\exists h \in H : \hat{R}(h)=0 \wedge R(h) > \varepsilon) \leq k(1-\varepsilon)^N$$
$$\leq |H|(1-\varepsilon)^N$$
$$\leq |H| e^{-\varepsilon N} \quad \leq \delta$$

Find N:

$$|H| \tfrac{1}{\delta} \leq e^{\varepsilon N}$$
$$\log |H| + \log \tfrac{1}{\delta} \leq \varepsilon N$$
$$N \geq \tfrac{1}{\varepsilon}\left(\log |H| + \log \tfrac{1}{\delta}\right)$$

Then with probability $\leq \delta$,
$\exists h \in H, R(h) > \varepsilon \wedge \hat{R}(h)=0$

i.e. with probability $> 1-\delta$,
$\forall h \in H, R(h) > \varepsilon \Rightarrow \hat{R}(h) > 0$

$\Leftrightarrow \forall h \in H, \hat{R}(h)=0 \Rightarrow R(h) \leq \varepsilon$

- **PAC bounds for finite model classes**
  - Haussler bound
    $\forall h, \hat{R}(h)=0 \Rightarrow R(h) < \varepsilon = \frac{1}{N}\left(\ln |H| + \ln \frac{1}{\delta}\right)$
  - Hoeffding bound
    $\forall h, |R(h) - \hat{R}(h)| \leq \varepsilon = \sqrt{\frac{1}{2N}\left(\ln |H| + \ln \frac{2}{\delta}\right)}$

- **Bias-variance tradeoff**
  - $P(|R(h) - \hat{R}(h)| \geq \varepsilon) \leq 2|H| e^{-2m\varepsilon^2} \leq \delta$
  - Equivalently, with probability $\geq 1-\delta$,

$$R(h) \leq \underbrace{\hat{R}(h)}_{\text{bias}} + \underbrace{\left(\frac{\ln |H| + \ln \frac{2}{\delta}}{2m}\right)^{1/2}}_{\text{variance}}$$

- **# decision trees depth k**
  - $H_k = $ # binary decision trees depth k, $H_0 = 2$, $H_k = n\, H_{k-1} H_{k-1}$
  - $L_k = \log_2 H_k$, $L_0 = 1$, $L_k = \log_2 n + 2L_{k-1} \xrightarrow{\text{solve}} L_k = (2^k - 1)(1 + \log_2 n) + 1$

Lecture 11/03 cont.

- **Plug into PAC bound**

  $m \geq \frac{\ln 2}{2\varepsilon^2}\left((2^k-1)(1+\log_2 n)+1+\log_2 \frac{2}{\delta}\right)$

  - Bad, exponential in k
  - But m data points $\Rightarrow$ at most m leaves

- **# decision trees with k leaves**
  - H_k = # binary decision trees with k leaves
  - $H_1 = 2$, $H_k = n\sum_{i=1}^{k-1} H_i H_{k-i} = n^{k-1} C_{k-1}$
  - (Stirling) $H_k \leq n^{k-1} 2^{2k-1}$

- **Comparison**
  - Depth k : $\log_2 H_k \leq (k-1)\log_2 n + 2k - 1$ — linear in k
  - k leaves : $\log_2 H_k = (2^k-1)(1+\log_2 n)+1$ — exponential in k

- **Shattering**
  - $H[S]$ = set of splittings of dataset S using concepts from H
  - H shatters S if $|H[S]| = 2^{|S|}$
  - VC-dimension of hypothesis space H is the cardinality of the largest set S that can be shattered by H
    - $VC(H) = \max\{|S| : H \text{ shatters } S\}$
    - If arbitrarily large finite sets can be shattered by H then $VC(H) = \infty$

- **To show $VC(H) = d$,**
  - Show $\exists S$, $|S| = d$, H shatters S
  - Show $\nexists S$, $|S| = d+1$, H shatters S

- Fact. H finite $\Rightarrow VC(H) \leq \log|H|$

- **SLT Corollaries**

	Realizable	Agnostic
Finite $\lvert H\rvert$	For $\delta>0$, $pr \geq 1-\delta$, $\forall h: \hat{R}(h)=0$, $R(h) \leq \frac{1}{N}\left[\ln\lvert H\rvert + \ln\frac{1}{\delta}\right]$	For $\delta>0$, $pr \geq 1-\delta$, $\forall h$, $R(h) \leq \hat{R}(h) + \sqrt{\frac{1}{2N}\left[\ln\lvert H\rvert + \ln\frac{2}{\delta}\right]}$
Infinite $\lvert H\rvert$	For $\delta>0$, $pr \geq 1-\delta$, $\forall h: \hat{R}(h)=0$, $R(h) \leq O\left(\frac{1}{N}\left[VC(H)\ln\left(\frac{N}{VC(H)}\right) + \ln\left(\frac{1}{\delta}\right)\right]\right)$	For $\delta>0$, $pr \geq 1-\delta$, $\forall h$, $R(h) \leq \hat{R}(h) + O\left(\sqrt{\frac{1}{N}\left[VC(H) + \ln\frac{1}{\delta}\right]}\right)$

Lecture 11/08 Bayesian Networks

Bayesian networks

- Directed acyclic graph, nodes = random variables, edges = dependency assumptions
- $p(x_1, \dots, x_n) = \prod_{i=1}^{n} p(x_i \mid \text{parent}(x_i))$

$$p(x_1, x_2, x_3) = P(x_1)\, P(x_2 \mid x_1)\, P(x_3 \mid x_1) = \begin{tikzcd} & x_1 \arrow[dl] \arrow[dr] & \\ x_2 & & x_3 \end{tikzcd}$$

- Middle ground between full independence and full dependence

Construction

① Identify the random variables

② Determine the conditional dependencies

- Select an ordering of variables ← changes network substantially!
- Add them one at a time
- For each new var X, select minimal subset of nodes as parents such that X is independent from all other nodes in the current network given its parents

③ Populate the conditional probability tables

- Using density estimation

Markov blanket

- All parents, children, co-parents of children
- In a Bayesian network a variable is conditionally independent of all other variables given its Markov blanket

$$\begin{tikzcd} B \arrow[dr] & & E \arrow[dl] \\ & A \arrow[dl] \arrow[dr] & \\ J & & M \end{tikzcd}$$

B's markov blanket: E, A

A's markov blanket: B, E, J, M

d-separation

- When are two variables independent of each other?
- x and y are d-separated given set of variables Z (possibly empty) if x and y are conditionally independent given Z
- $I(x, y \mid Z)$ denotes the above conditional independence
- Variables that are not d-connected are d-separated
- Rules

  ① If Z empty, x and y are d-connected if ∃ path between them without collider

  ② x and y are d-connected given Z if ∃ path between them without collider nor member of Z

  ③ If Z contains collider or one of its descendants, and no other node from Z on the path, then if a path between x and y contains this node they are d-connected

$$\begin{tikzcd} X \arrow[r] & \overset{\text{collider}}{\bigcirc} & Y \arrow[l] \end{tikzcd}$$

Causality warning

$$\begin{tikzcd} A \arrow[r] & B \end{tikzcd} = P(A)\, P(B \mid A) = P(A \cap B) = P(B)\, P(A \mid B) = \begin{tikzcd} B \arrow[r] & A \end{tikzcd}$$

Lecture 11/10 Bayesian Network Inference

· Inference

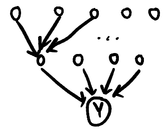
- Enumeration
- Stochastic inference
- Variable elimination
- Tree conversion

· Enumeration

- Sum over all possibilities for other vars
- Exponential runtime
  - Reusing calculations (like dynamic programming) may help
  - But # possible assignments still exponential in unobserved vars

· General querying of Bayesian networks is NP-complete

- Reduction from 3SAT
- Three layer



} $P(x_i=1) = 0.5$

} $P(x_i=1) = x_1 \vee x_2 \vee x_3$

} $P(Y=1) = x_1 \wedge x_2 \wedge x_3 \wedge x_4$

$P(Y=1) \xrightarrow{>0}$ satisfiable

$\xrightarrow{=0}$ not satisfiable

· Stochastic inference

- Sample the joint distribution to obtain possible instances
  1. Sample free variables ($\exists$ since acyclic)
  2. For every other variable,
     if all parents sampled,
     sample based on conditional distribution
- Example, $P(B|J,\neg M)$
  - N = # samples
  - N_C = # samples where $J, \neg M$ holds (condition)
  - N_B = # samples where $B, J, \neg M$ holds (joint)
  - $N_C/N \approx P(J,\neg M)$
  - $N_B/N \approx P(B,J,\neg M)$
  - So $P(B|J,\neg M) = \frac{P(B,J,\neg M)}{P(J,\neg M)} \approx \frac{N_B}{N_C}$
- Problem: only samples where condition holds are used, huge waste
  - We can fix that?
  - But problem: fixing changes distribution, consider

$$\begin{tikzcd} \textcircled{A} \arrow[r] & \textcircled{B} \end{tikzcd}$$

$P(B=1|A=1) = 0.001$

$P(B=0|A=1) = 0.999$

$P(B=1|A=0) = 0.5$

Lecture 11/10 cont.

- **Weighted sampling**, $P(B|J,\neg M)$ example
  - Always $J \leftarrow 1$, $M \leftarrow 0$
  - Sample as before
  - Let $w = P(v(B), v(E), v(A), J, \neg M)$ be the weight of the sample
  - Alg
    - $N_B, N_c \leftarrow 0, 0$
    - Sample joint w/ fixed J, M.
    - Compute weight w of sample.
    - $N_c \leftarrow N_c + w$
    - If $B=1$, $N_B \leftarrow N_B + w$
    - After many iterations, output $P(B|J,\neg M) \approx N_B/N_c$

- **Variable elimination**
  - Store and reuse $P(M|a)$, $P(J|a)$

$$\begin{tikzcd}
B \arrow[dr] & & E \arrow[dl] \\
& A \arrow[dl] \arrow[dr] & \\
J & & M
\end{tikzcd}$$

  - Alg
    - Let e = evidence (known variables)
    - Let vars = conditional probabilities derived from network in bottom up reverse order
    - For var in vars

```
    factors ← make_factors(var, e)
    if var hidden, create new factor by summing out var
```

    - Compute the product of all factors
    - Normalize
  - Complexity now depends on highest in-degree

- **Polytree**
  - No two nodes have more than one directed path between them
  - Conversion by clustering, $\exists$ alg linear in number of nodes
  - But conversion can result in exponential increase in CPT size

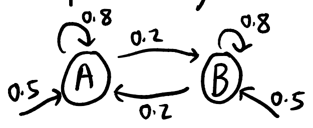
Lecture 11/15 Hidden Markov Models

- Limitations of Bayesian networks
  - Cannot account for temporal/sequence models
  - Must be DAGs

- Hidden Markov models HMM
  - Model set of observations with a set of hidden states
  - Hidden states generate observations
  - Hidden states transition to other hidden states
  - Fix a set of states $\{s_1, \dots, s_n\}$
    - In each time point we are in exactly one of these states q_t
  - Fix π_i the probability we start at state s_i
  - Learn transition probability model $P(q_t = s_i \mid q_{t-1} = s_j)$
    - Markov property $P(q_{t+1} = s_i \mid q_t = s_j) = P(q_{t+1} = s_i \mid q_t = s_j, q_{t-1} = s_k)$
  - Fix a set of possible outputs Σ (the observations)
    - At time t, emit symbol $\sigma \in \Sigma$
  - Learn an emission probability model $p(o_t = \sigma \mid s_i)$

- HMM inference
  - Cannot look at observations ⇒ compute $P(Q)$, $P(q_t = s_i)$
  - Have observations, only care about last state ⇒ compute $P(Q|O)$, $P(q_t = s_i | O)$
  - Care about entire path ⇒ compute $\text{argmax}_Q \ P(Q|O)$

- Example dice game



  - Let Q = any path ending in A
    - $P(Q) = P(q_1, q_2, \dots, q_{t-1}, A) = P(A | q_1, \dots, q_{t-1}) P(q_1, \dots, q_{t-1})$
      $= P(A | q_{t-1}) P(q_1, \dots, q_t)$
      $= P(A | q_{t-1}) \cdots P(q_2 | q_1) P(q_1)$
    - $P(q_t = A) = \underset{\uparrow \, 2^{t-1} \text{ terms!}}{\Sigma P(Q)}$
    - Let $p_t(i) = p(q_t = s_i)$
      - $p_1(i) = \pi_i$
      - $p_t(i) = \sum_j P(q_t = s_i | q_{t-1} = s_j) p_{t-1}(j)$
      - Dynamic programming $O(n^2 t)$

  } Computing $P(Q)$ efficiently

- Limit theorem for Markov transitions
  - If we see no observations, transition matrix strictly positive, then $\lim_{k \to \infty} (P^k)_{ij} = \theta_j$, i.e., starting point doesn't matter

Lecture 11/15 cont.

- Computing $P(Q|O)$ and $P(q_t=s_i|O)$
  - Transition probability $a_{ji}=P(q_t=s_i|q_{t-1}=s_j)$
  - Emission probability $b_i(o_t)=P(o_t|s_i)$
  - $P(Q|O)=\frac{P(O|Q)P(Q)}{P(O)}=\frac{P(o_1|q_1)P(o_2|q_2)\cdots P(o_t|q_t)P(q_1)P(q_2|q_1)\cdots P(q_t|q_{t-1})}{P(O)}$
  - To compute $P(O)$,
    let $\alpha_t(i)=P(o_1,o_2,\dots,o_t\wedge q_t=s_i)$
  - Then $\alpha_{t+1}(i)=\sum_j P(o_1\dots o_t\wedge q_t=s_j\wedge o_{t+1}\wedge q_{t+1}=s_i)$
    $=\sum_j P(o_{t+1}\wedge q_{t+1}=s_i|o_1\dots o_t\wedge q_t=s_j)P(o_1\dots o_t\wedge q_t=s_j)$
    $=\sum_j P(o_{t+1}|q_{t+1}=s_i)P(q_{t+1}=s_i|q_t=s_j)\alpha_t(j)$
    $=\sum_j b_i(o_{t+1})a_{ji}\alpha_t(j)$
  - So $P(O)=\sum_i \alpha_t(i)$
    Note $P(q_t=s_i|o_1,o_2,\dots,o_t)=\frac{\alpha_t(i)}{\sum_j\alpha_t(j)}$
  - Complexity
    - $P(Q)=O(t)$
    - $P(O|Q)=O(t)$
    - $P(O)=O(n^2t)$

- Computing $P(Q^*|O)=\text{argmax}_Q P(Q|O)$
  - Let $\delta_t(i)=\max_{q_1,\dots,q_{t-1}} P(q_1,\dots,q_{t-1}\wedge q_t=s_i\wedge o_1\dots o_t)$ "$\text{path}_{1\mapsto t}$ that ends in S_t, outputs $o_1\dots o_t$"
    Then $\delta_1(i)=\pi_i\, b_i(o_1)$
    and $\delta_{t+1}(i)=\max_{q_1,\dots,q_t} P(q_1,\dots,q_t\wedge q_{t+1}=s_i\wedge o_1\dots o_{t+1})$
    $=\max_j \delta_t(j)P(q_{t+1}=s_i|q_t=s_j)P(o_{t+1}|q_{t+1}=s_i)$
    $=\max_j \delta_t(j)a_{ji}b_i(o_{t+1})$

- Viterbi Algorithm
  $P(Q^*|O)=\text{argmax}_Q P(Q|O)$
  $=$ path defined by $\text{argmax}_j\ \delta_t(j)$

Lecture 11/17 Learning HMMs

- We will learn transition and emission models
  - Set of states usually domain knowledge

- Initial probabilities
  $\pi^*=\text{argmax}_\pi \prod_k \pi(q_1)\prod_{t=2}^T P(q_t|q_{t-1})$ $k=\#$ sequences available from training
  $=\text{argmax}_\pi \prod_k \pi(q_1)$

- Transition probabilities
  $a^*=\text{argmax}_a \prod_k \pi(q_1)\prod_{t=2}^T P(q_t|q_{t-1})$
  $=\text{argmax}_a \prod_{t=2}^T P(q_t|q_{t-1})$

- Above assumes we have states, but states usually not known $\Rightarrow$ EM

Lecture 11/17 cont.

- Forward-Backward
  - Forward $\alpha_t(i) = P(O_1 \wedge \dots \wedge O_t \wedge q_t = i)$
  - Backward $\beta_t(i) = P(O_{t+1} \wedge \dots \wedge O_T \mid q_t = s_i)$
    $= \sum_j \alpha_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$
  - $P(q_t = s_i \mid O_1, \dots, O_T) = \frac{\alpha_t(i)\beta_t(i)}{\sum_j \alpha_t(j)\beta_t(j)} =: S_t(i)$
  - $P(q_t = s_i, q_{t+1} = s_j \mid O_1, \dots, O_T) = \frac{\alpha_t(i) P(q_{t+1} = s_j \mid q_t = s_i) P(O_{t+1} \mid q_{t+1} = s_j) \beta_{t+1}(j)}{\sum_k \alpha_t(k) \beta_t(k)} =: S_t(i,j)$

- E step
  - Compute $S_t(i)$ and $S_t(i,j)$ for all t, i, j $\quad 1 \le t \le n,\ 1 \le i \le k,\ 2 \le j \le k$

- M step
  - Compute emission probabilities
    - Let $B_k(j) = \sum_{t \mid O_t = j} S_t(k)$
      Then $b_k(j) = \frac{B_k(j)}{\sum_i B_k(i)}$
  - Compute transition probabilities
    $a_{ij} = \frac{\hat{n}(i,j)}{\sum_k \hat{n}(i,k)}$
    where $\hat{n}(i,j) = \sum_t S_t(i,j)$

- Complete EM (Baum Welch)
  - Input
    - Observations $O_1, \dots, O_T$
    - Number of states, model
  - Alg
    ```
    ① Guess initial transition and emission parameters
    ② Until convergence
          Compute E
          Compute M
    ③ Output complete model
    ```

- Advanced HMMs
  - Factorial HMMs
  - Input-output HMMs
  - Dynamic Bayesian Networks

- Factorial HMMs
  - Decouple independent states
  - M same as HMM
  - E hard, can be exponential in number of states, usually use sampling (Monte Carlo)

- Input-output model
  - Static input: input layer only affects transition probabilities
    $p(Q,O \mid M) = \pi(q_1)\, p(O_1 \mid q_1) \prod_t p(q_t \mid q_{t-1}, R)\, p(O_t \mid q_t)$ where R is input

Lecture 11/17 cont.

- Input-output HMM learning
  - Learn new transition table for each R (note emissions same)
  - Output can also depend on R
- Dynamic Bayesian Networks
  - BN doesn't allow for feedback
  - $P(X) = \prod_i p(x_i | Pa(x_i))$
  - Repeat the network over time slices

Lecture 11/22 Markov Decision Processes

- HMM and MDP

HMM
$$\begin{tikzcd} s_1 \arrow[r] \arrow[d] & s_2 \arrow[r] \arrow[d] & s_3 \arrow[r] \arrow[d] & \cdots \\ o_1 & o_2 & o_3 & \end{tikzcd}$$

MDP
$$\begin{tikzcd} a_1 \arrow[dr] & a_2 \arrow[dr] & a_3 \arrow[dr] & \\ s_1 \arrow[r] & s_2 \arrow[r] & s_3 \arrow[r] & \cdots \end{tikzcd}$$

- Minimax
  $V(s) = \max_a V(s')$ where $s' = result(s,a)$
  $a = \operatorname{argmax}_a V(s')$ where $s' = result(s,a)$

- Chance node notation
  $V(s) = \sum_{s'} P(s') V(s')$

- Expectimax : minimax, replace min with chance
  $V(s) = \max_a \sum_{s'} [P(s'|s,a) V(s')]$

- Markov Decision Process
  - Set of states $s \in S$
  - Set of actions $a \in A$
  - Transition function $T(s,a,s')$, also called model or dynamics
    - $P(s'|s,a)$
  - Reward function $R(s,a,s')$, sometimes $R(s)$ or $R(s')$
  - Start state
  - Maybe terminal state
  - Nondeterministic search problem
    - Solve with e.g., expectimax

- Markov assumption
  $P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \dots, S_0 = s_0)$
  $= P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$

Lecture 11/22 cont.

- Policies
  - $\pi : S \to A$
  - π^* optimal if maximizes expected utility
  - Explicit policy defines reflex action
  - Expectimax does NOT compute entire policies
    - Only finds actions for a single state

- Recursive Expectimax
  - $V(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + V(s')]$

- Discounting
  - Values of rewards decay exponentially, multiply γ per level
  - Sooner rewards
    - Probably higher utility than later rewards
    - Helps convergence

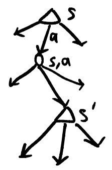
- What if the game lasts forever?
  - Finite horizon, up to T steps
    - Gives nonstationary policies
  - Discounting $0 < \gamma < 1$
    - $V([r_0, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \le \frac{R_{max}}{1-\gamma}$
    - Smaller γ means smaller horizon, short term focus

- Value iteration
  ① $V_0(s) = 0$
  ② Repeat until convergence, expectimax from each state
  $V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$
  $O(S^2A)$ complexity each iteration

- Convergence of V_k
  - Case 1: tree has max depth M, then V_M has actual untruncated values
  - Case 2: discount <1, then
    view V_k, V_{k+1} as almost identical depth k+1 expectimax result trees
    except V_k's bottom is all 0's, V_{k+1}'s bottom has actual rewards
    Last layer at best R_{max} at worst R_{min}
    But discounting $\Rightarrow$ at most $\gamma^k \max |R|$ apart $\Rightarrow$ converge as k increases

---

Lecture 11/29 except it was 2020 recording

- 

s : state
(s,a) : q-state
(s,a,s') : transition

$V^*(s)$ = expected utility starting in s and acting optimally
$Q^*(s,a)$ = expected utility starting in s, taking action a, then acting optimally
$\pi^*(s)$ = optimal action from state s

Lecture 11/29 cont.

- **Recursive value definitions**
  - $V^*(s) = \max_a Q^*(s,a)$
  - $Q^*(s,a) = \sum_{s'} T(s,a,s')\,[R(s,a,s') + \gamma V^*(s')]$
  - $V^*(s) = \max_a \sum_{s'} T(s,a,s')\,[R(s,a,s') + \gamma V^*(s')]$

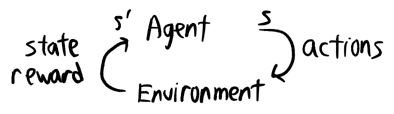
- **Computing actions**
  - Given Q-values, just pick largest
  - Given values, perform policy extraction with mini expectimax
    $\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')\,[R(s,a,s') + \gamma V^*(s')]$

- **Solving MDPs**, two approaches
  - **Value iteration + policy extraction**
    VI ① $V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\,[R(s,a,s') + \gamma V_k(s')]$ $\forall s$ until convergence
    PE ② $\pi_V(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)\,[R(s,a,s') + \gamma V(s')]$
  - **Policy iteration = policy evaluation + policy improvement**
    PE ① $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))\,[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')]$ $\forall s$ until convergence
    PI ② $\pi_{new}(s) = \arg\max_a \sum_{s'} P(s'|s,a)\,[R(s,a,s') + \gamma V^{\pi_{old}}(s')]$ $\forall s$
    Repeat ①, ② until policy converges

- **Summary**

Standard expectimax	$V(s) = \max_a \sum_{s'} P(s'\|s,a)\,V(s')$
Bellman equations	$V^*(s) = \max_a \sum_{s'} P(s'\|s,a)\,[R(s,a,s') + \gamma V^*(s')]$
Value iteration	$V_{k+1}(s) = \max_a \sum_{s'} P(s'\|s,a)\,[R(s,a,s') + \gamma V_k(s')]$
Q-iteration	$Q_{k+1}(s,a) = \sum_{s'} P(s'\|s,a)\,[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$
Policy extraction	$\pi_V(s) = \arg\max_a \sum_{s'} P(s'\|s,a)\,[R(s,a,s') + \gamma V(s')]$
Policy evaluation	$V^{\pi}_{k+1}(s) = \sum_{s'} P(s'\|s,\pi(s))\,[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')]$

- **Reinforcement learning**
  - Note that **solving MDPs is all offline**
  - What if **transition and reward functions not known**, must be learned?
  - Concepts
    - **Exploration** : try unknown actions to get information
    - **Exploitation** : eventually use what you know
    - **Regret** : mistakes will be made
    - **Sampling** : must try repeatedly because of chance
    - **Difficulty** : learning can be much harder than solving known MDP
  - Framework



All learning based on observed samples of outcomes

Lecture 11/29 cont.

- Model-Based Learning
  - Learn approx model based on experiences
  - Solve for values as if learned model correct
  - ① Learn empirical MDP model
    - count outcomes for each (s,a)
    - normalize to estimate $\tilde{T}(s,a,s')$
    - discover each $\tilde{R}(s,a,s')$ when experiencing (s,a,s')
  - ② Solve the learned MDP

- Sample based policy evaluation

  $V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi,s') + \gamma V_k^{\pi}(s')]$

  Instead, sample and average

  $sample_i = R(s,\pi(s),s_i') + \gamma V_k^{\pi}(s_i')$

  $V_{k+1}^{\pi}(s) = \frac{1}{n}\sum_i sample_i$

- Temporal difference learning
  - Learn from every (s,a,s',r) experience
  - Likely s' outcomes contribute updates more often
  - Note policy is fixed, just doing evaluation. We shift values.
  - Components
    - Sample $V(s)$ $sample = r + \gamma V^{\pi}(s')$
    - Update $V(s)$ $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\, sample$
    - Same update $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

      or $V^{\pi}(s) \leftarrow V^{\pi}(s) - \alpha \nabla error$ where $\nabla error = \frac{1}{2}(sample - V^{\pi}(s))^2$

- Q-value learning
  - TD value learning above: model free policy evaluation, but how to get a policy?
  - Learn Q-values instead of values ⇒ then action selection is model free too
  - Q-value iteration
    - $Q_0(s,a) = 0$
    - $Q_{k+1}(s,a) = \sum_{s'} \underset{\text{not known}}{T(s,a,s')}[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$
    - Compute average as we go
    - $Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$
    - But need to average over results from (s,a)
    - So $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]$

- Q-learning properties
  - Off-policy learning
    - Q-learning converges to optimal policy even if acting suboptimally
  - Caveats
    - Must explore enough
    - Must eventually make learning rate small enough but not decrease it too quickly
  - But in the limit, doesn't matter how actions are selected

Lecture 11/29 cont.

- Exploration vs Exploitation
  - ε-greedy exploration
    - Each time step, act randomly with probability ε, act according to current policy with probability $1-\varepsilon$
  - But random actions problematic
    - Eventually explores everything but keeps thrashing around when learning done
    - Fix: lower ε over time
    - Another fix: exploration functions

- Feature-based learning
  - State space huge $\Rightarrow$ use features instead
  - $V_w(s) = \sum_{i=1}^{M} w_i f_i(s)$
  - $Q_w(s,a) = \sum_{i=1}^{M} w_i f_i(s,a)$
  - But states which share features may be very different in value
  - To update the linear value function,
    $w_i \leftarrow w_i + \alpha [R(s,a,s') + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)] \underbrace{\frac{\partial Q_w(s,a)}{\partial w_i}}_{f_i(s,a)}$
  - Qualitatively
    - Pleasant surprise : $\uparrow$weight +ve features, $\downarrow$weight -ve ones
    - Unpleasant surprise : $\downarrow$weight +ve features, $\uparrow$weight -ve ones

- Approximate Q-learning
  - $Q_w(s,a) = \sum_{i=1}^{M} w_i f_i(s,a)$
  - transition = (s,a,s',r)
  - diff $= [r + \gamma \max_{a'} Q(s',a')] - Q(s,a)$
  - $Q(s,a) \leftarrow Q(s,a) + \alpha\,\text{diff}$ exact Q
  - $w_i \leftarrow w_i + \alpha\,\text{diff}\, f_i(s,a)$ approx Q
  - Formal justification: online least squares

- Minimizing error
  - $\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$
  - $\frac{\partial\, \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$
  - $w_m \leftarrow w_m + \alpha\left(y - \sum_k w_k f_k(x)\right) f_m(x)$

Probability

$P(A \cup B) = P(A) + P(B) - P(A \cap B)$ axiom

$P(A|B,C) = P(A|C) \Leftrightarrow (A \perp\!\!\!\perp B|C) \Leftarrow$ A and B conditionally independent given C

$P(A,B) = P(A|B)P(B)$ chain rule

$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ Bayes rule

$P(\theta|D) \propto P(D|\theta)P(\theta)$ posterior $\propto$ likelihood $\cdot$ prior

$\hat{\theta}_{MLE} = \operatorname{argmax}_\theta P(D|\theta)$ MLE

$\hat{\theta}_{MAP} = \operatorname{argmax}_\theta P(D|\theta)P(\theta)$ MAP

Classifiers

- Instance-based : eg kNN use observations directly
- Generative : eg Bayesian network generative statistical model, $P(X|Y)$
- Discriminative : eg decision tree directly estimate decision boundary, no $P(X|Y)$

kNN: majority vote on k "closest" points

Naive Bayes: $P(X|y) = \prod_{i=1}^{N} \prod_j p_j(x_j^{(i)}|y)$

Note $P(y|X)$ needs $P(Y)$

Pseudocounts: add $\vec{x} = \vec{1}$, $\vec{x} = \vec{0}$ samples

$H(X) = \sum_c -P(X=c)\log_2 P(X=c)$ **entropy**

$H(Y|X) = \sum_i P(X=i)H(Y|X=i)$ conditional entropy

$IG(Y|X) = H(Y) - H(Y|X)$ information gain, ≥ 0 by Jensen

$f(tx_1 + (1-t)x_2) \leq t f(x_1) + (1-t)f(x_2)$ f convex, $0 \leq t \leq 1$ Jensen inequality

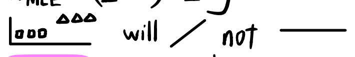
Decision tree ID3: split on attribute maximizing information gain

Bagging: reduce variance by randomly drawing datasets with replacement

Random Forest: bagging + use subset of features at each tree node

Linear Regression: $y = \sum_{j=0}^{n} w_j \phi_j(x)$ ← linear coefficients

$\hat{w}_{MLE} = (\Phi^T \Phi)^{-1} \Phi^T y$



Sigmoid $g(h) = \frac{1}{1+e^{-h}}$

- $P(y=0|X;\theta) = g(w^T x) = \frac{1}{1+e^{w^T x}}$
- $P(y=1|X;\theta) = 1 - g(w^T x)$

Logistic regression: no closed form, but concave so gradient ascent

- $LL(y|X;w) = \sum_{i=1}^{N} y_i \ln(1-g(X_i;w)) + (1-y_i)\ln(g(X_i;w))$

Regularization : L_2 : $\min \sum w_i^2$; L_1 : $\min \sum_i |w_i|$

SVM

$$\min_w \frac{w^T w}{2} + \underbrace{\sum_{i=1}^{n} C\varepsilon_i}^{\text{non-linearly separable}}$$

subject to

$\forall x_i$ in class +1, $w^T x + b \geq 1 - \varepsilon_i$

$\forall x_i$ in class −1, $w^T x + b \leq -1 + \varepsilon_i$

$\forall i$, $\varepsilon_i \geq 0$

As $C \to \infty$, hard margin, forced linear sep
$C \to 0$, soft margin

Lagrange multiplier $\min_x x^2$ st $x \geq b$ $\rightarrow$ $\min_x \max_\alpha x^2 - \alpha(x-b)$ st $\alpha \geq 0$

True error = bias + variance

AdaBoost: iteratively reweight inputs, more weight to hard predictions

- Boosting increases (confidence) margin even after train error = 0

Distances ① $D(A,B) = D(B,A)$ ② $D(A,A) = 0$ ③ $D(A,B) = 0$ iff $A=B$ ④ $D(A,B) \leq D(A,C) + D(C,B)$

Minkowski $d(\vec{x},\vec{y}) = \left(\sum_{i=1}^{p} |x_i - y_i|^r\right)^{1/r}$

Clustering: hierarchical (agglomerative / divisive), partition (k-means / mixture model)

Gaussian Mixture Model $p(x) = \sum_{k=1}^{K} P(x|z_k=1)P(z_k=1)$ where $P(x|z_k=1) \sim N(\mu_k, \Sigma_k)$

EM

- E : $\mathbb{E}_{z|x,\theta^{(t)}}[z_k^{(i)}] = p(z_k^{(i)}=1 | x^{(i)}, \theta^{(t)})$
- M : $\operatorname{argmax}_\theta \mathbb{E}_{z|x,\theta^{(t)}}[\ell_c(\theta|D_c)]$

GMM + hard assignment = k-Means

Calculus

$$\frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & & \frac{\partial y_1}{\partial x_M} \\ \frac{\partial y_2}{\partial x_1} & \cdots & \vdots \\ \vdots & & \\ \frac{\partial y_N}{\partial x_1} & & \frac{\partial y_N}{\partial x_M} \end{bmatrix},$$

numerator $y\downarrow$ $x\rightarrow$; denominator is transpose

	Numerator layout	Denominator layout
$\frac{\partial}{\partial \vec{v}} \vec{v}$	I_N	I_N
$\frac{\partial}{\partial \vec{v}} \vec{v}^T$	I_N	I_N
$\frac{\partial}{\partial \vec{v}} t\vec{v}$	tI_N	tI_N
$\frac{\partial}{\partial \vec{u}} \vec{u}^T\vec{v}$	$\vec{v}^T$	$\vec{v}$
$\frac{\partial}{\partial \vec{v}} \vec{u}^T\vec{v}$	$\vec{u}^T$	$\vec{u}$
$\frac{\partial}{\partial \vec{v}} \vec{v}^T\vec{v}$	$2\vec{v}^T$	$2\vec{v}$
$\frac{\partial}{\partial \vec{v}} \vec{v}^T A\vec{v}$	$\vec{v}^T(A+A^T)$	$(A+A^T)\vec{v}$
$\frac{\partial}{\partial \vec{v}} A\vec{v}$	A	A^T
$\frac{\partial}{\partial \vec{v}} \vec{v}^T A$	A^T	A
$\frac{d}{dt} f(g(t),h(t))$	$\frac{\partial f}{\partial g}\frac{\partial g}{\partial t} + \frac{\partial f}{\partial h}\frac{\partial h}{\partial t}$	← same
$\frac{d}{dt} f(g_1(t),\dots,g_n(t))$	$\sum_{i=1}^{n} \frac{\partial f}{\partial g_i}\frac{\partial g_i}{\partial t}$	← same
$\frac{d}{dt} f(\vec{g}(t))$	$\frac{\partial f}{\partial \vec{g}}\frac{\partial \vec{g}}{\partial t}$	$\frac{\partial \vec{g}}{\partial t}\frac{\partial f}{\partial \vec{g}}$
$\frac{d}{d\vec{v}} f(\vec{g}(\vec{v}))$	$\frac{\partial f}{\partial \vec{g}}\frac{\partial \vec{g}}{\partial \vec{v}}$	$\frac{\partial \vec{g}}{\partial \vec{v}}\frac{\partial f}{\partial \vec{g}}$
$\frac{\partial}{\partial \vec{v}} f(\vec{g}(\vec{v}),\vec{h}(\vec{v}))$	$\frac{\partial f}{\partial \vec{g}}\frac{\partial \vec{g}}{\partial \vec{v}} + \frac{\partial f}{\partial \vec{h}}\frac{\partial \vec{h}}{\partial \vec{v}}$	$\frac{\partial \vec{g}}{\partial \vec{v}}\frac{\partial f}{\partial \vec{g}} + \frac{\partial \vec{h}}{\partial \vec{v}}\frac{\partial f}{\partial \vec{h}}$

Learning Theory, $R(h)$ true error $\hat{R}(h)$ train error

	Realizable	Agnostic
Finite $\lvert H\rvert$ $\propto \frac{1}{\varepsilon}, \frac{1}{2}\varepsilon \Rightarrow 2\times N$ $\propto \frac{1}{\varepsilon^2}, \frac{1}{2}\varepsilon \Rightarrow 4\times N$ $\propto \log\lvert H\rvert, 4\lvert H\rvert \Rightarrow 2\times N$	$N \geq \frac{1}{\varepsilon}\left[\log\lvert H\rvert + \log\frac{1}{\delta}\right]$ wp $1-\delta$, $\hat{R}(h)=0 \Rightarrow R(h)\leq\varepsilon$ $\neg\exists\delta>0$, wp $\geq 1-\delta$, $\hat{R}(h)=0 \Rightarrow$ $R(h) \leq \frac{1}{N}\left(\ln\lvert H\rvert + \ln\frac{1}{\delta}\right)$	$N \geq \frac{1}{2\varepsilon^2}\log\lvert H\rvert + \log\frac{2}{\delta}$ wp $1-\delta$, $\lvert R(h)-\hat{R}(h)\rvert \leq \varepsilon$ $\neg\exists\delta>0$, wp $\geq 1-\delta$, $R(h) \leq \hat{R}(h) + \sqrt{\frac{1}{2N}\left(\ln\lvert H\rvert + \ln\left(\frac{2}{\delta}\right)\right)}$
Infinite $\lvert H\rvert$	$N = O\left(\frac{1}{\varepsilon}\left[VC(H)\log\frac{1}{\varepsilon} + \log\frac{1}{\delta}\right]\right)$ wp $1-\delta$, $\hat{R}(h)=0 \Rightarrow R(h)\leq\varepsilon$ $\neg\exists\delta>0$, wp $\geq 1-\delta$, $\hat{R}(h)=0 \Rightarrow$ $R(h) \leq O\left(\frac{1}{N}\left(VC(H)\ln\frac{N}{VC(H)} + \ln\frac{1}{\delta}\right)\right)$	$N = O\left(\frac{1}{\varepsilon^2}\left[VC(H) + \log\frac{1}{\delta}\right]\right)$ wp $1-\delta$, $\lvert R(h)-\hat{R}(h)\rvert \leq \varepsilon$ $\neg\exists\delta>0$, wp $\geq 1-\delta$, $R(h) \leq \hat{R}(h) + O\sqrt{\frac{1}{N}\left(VC(H)+\ln\frac{1}{\delta}\right)}$

PCA

- Always center data by subtracting sample mean
  - $\mu = \frac{1}{N}\sum_{i=1}^{N}\vec{x}^{(i)} = \vec{0}$, $\Sigma = \frac{1}{N}X^TX$

① Center each axis $\rightarrow X, X_{test}$
② $V =$ eigenvectors (X^TX)
③ Keep top k V_k
④ $Z_{test} = X_{test}V_k$

Projection $(x,\vec{v}) = \frac{x^T\vec{v}}{\|\vec{v}\|_2}\vec{v}$

Reconstruction Error $\quad v^* = \underset{\vec{v}:\|\vec{v}\|_2=1}{\text{argmax}} \sum_{i=1}^{N}\left\|\vec{x}^{(i)} - (\vec{v}^T\vec{x}^{(i)})\vec{v}\right\|_2^2$

Variance of Projection $\quad v^* = \underset{\vec{v}:\|\vec{v}\|_2=1}{\text{argmax}} \sum_{i=1}^{N}\left(\vec{v}^T\vec{x}^{(i)}\right)^2$

NN

$$L_{out} = \left\lfloor \frac{L_{in} - \overset{\text{kernel}}{K} + 2\overset{\text{pad}}{P}}{\underset{\text{stride}}{S}} + 1 \right\rfloor$$

BN

- $P(x_1 \cdots x_n) = \prod_i P(x_i \mid Pa(x_i))$
- X, Y conditionally indep given Z if $P(X,Y|Z) = P(X|Z)P(Y|Z)$
- In BN, X conditionally indep of all other var given Markov blanket
- Markov Blanket = all parents, children, coparents of children
- d-connected rules

① $Z=\phi \Rightarrow X,Y$ d-connected if $\exists X \rightsquigarrow Y$ no collider
② X,Y d-connected given Z if $\exists X \rightsquigarrow Y$ no collider, no member of Z
③ If Z has collider or collider descendant, no other node from Z on path,
if $\exists X \rightsquigarrow Y$ has that node then d-connected

- d-separated $\Leftrightarrow$ not d-connected $\Leftrightarrow$ X,Y conditionally indep given Z

- Construction
  ① Identify the random variables
  ② Determine the conditional dependencies
    - Select an ordering of variables $\leftarrow$ changes network substantially!
    - Add them one at a time
    - For each new var X, select minimal subset of nodes as parents such that X is independent from all other nodes in the current network given its parents
  ③ Populate the conditional probability tables
    - Using density estimation

- Inference
  - Enumeration
  - Stochastic inference
  - Variable elimination
  - Tree conversion

HMM

- $P(q_1 q_2 \cdots A)$
  $= P(A \mid q_1 \cdots q_{t-1})P(q_1 \cdots q_{t-1})$
  $= P(A \mid q_{t-1})P(q_{t-1} \mid q_{t-2}) \cdots P(q_2 \mid q_1)P(q_1)$
- $P(q_t = s_i)$ i.e. $P_t(i)$
  $P_1(i) = \pi_i$, $P_t(i) = \sum_j P(q_t = s_i \mid q_{t-1} = s_j)P_{t-1}(j)$
- $P(Q|O)$, $P(q_t = s_i|O)$
  - $P(O|Q) = P(o_1|q_1) \cdots P(o_t|q_t)$
  - $P(Q) = P(q_t|q_{t-1}) \cdots P(q_2|q_1)P(q_1)$
  - $\alpha_t(i) = P(o_1 \wedge o_2 \wedge \ldots \wedge o_t \wedge q_t = s_i) = \sum_j b_i(O_t)a_{ji}\alpha_{t-1}(j)$
  - $a_{ji} = P(q_t = s_i \mid q_{t-1} = s_j)$
  - $b_i(O_t) = P(o_t \mid q_t = s_i)$
  - $P(O) = \sum_i \alpha_t(i)$
  - $P(q_t = s_i \mid o_1 \ldots o_t) = \frac{\alpha_t(i)}{\sum_j \alpha_t(j)}$
- $\text{argmax}_Q\, P(Q|O) =$ path of $\text{argmax}_j\, \delta_t(j)$
  - $\delta_t(i) = \max_{q_1 \cdots q_{t-1}} P(q_1 \cdots q_{t-1} \wedge q_t = s_i \wedge o_1 \cdots o_t)$
    $= \max_j \delta_{t-1}(j)a_{ji}b_i(O_t)$

RL

Standard expectimax	$V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$
Bellman equations	$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$
Value iteration	$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]$
Q-iteration	$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$
Policy extraction	$\pi_V(s) = \text{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
Policy evaluation	$V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')]$
Value (TD) learning	$V^{\pi}(s) = V^{\pi}(s) + \alpha[r + \gamma V^{\pi}(s') - V^{\pi}(s)]$
Q-learning	$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$