Supervised Algorithms

Regression

- Single and multivariate
- $h_{\mathbf{w}}(\mathbf{x_j}) = \mathbf{w}^\intercal \mathbf{x_j} = \sum_i w_i x_{j,i}$
- $\mathbf{w}^* = \text{argmin}_{\mathbf{w}} \sum_j L_2(y_j, \mathbf{w}^\intercal \mathbf{x_j})$
- Regularization γ to penalize complexity
- $Cost(h) = EmpLoss(h) + \gamma L_q(\mathbf{w})$
- Complexity: $L_q(\mathbf{w}) = \sum_i |w_i|^q$
- **Linear Regression**
  - Directly computable
- **Non-Linear Regression**
  - Differentiable, use *gradient descent*
- **Stepwise Regression** (Perceptron function)
  - Linearly separable data/Hard Threshold
  - $h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x})$
  - $Threshold(z) = 1$ if $z \geq 0, 0$ otherwise
  - Not differentiable - use perceptron learning
- **Logistic Regression** (Sigmoid Function)
  - Soft threshold
  - $Logistic(z) = \frac{1}{1+e^{-z}}$
  - $h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x})$
  - Differentiable, use *gradient descent*
  - Nice property: $g'(u) = g(u)(1 - g(u))$

Decision Trees (DT)

- *Analogy*: human decision-making paradigm
- Can represent any truth table
- *Inductive bias*: best attribute/dimension first
- Best = Highest information gain (entropy)
- Hypothesis complexity: # tree levels and nodes
- Discrete attributes: split point for each value
- Continuous attributes: establish split threshold
- Prune tree to eliminate overfitting
- **Iterative Dichotomiser** (ID3)

Instance Based

- Nonparametric: grows with # of examples
- *Inductive Bias*:
  - locality (near points are similar)
  - all dimensions are of equal importance
- Lazy learner/just-in-time learning
- Curse of Dimensionality, w/ N examples, d dim.:
  - $N = O(2^d)$ to maintain generalization accuracy.
- **K Nearest Neighbors** (KNN) Classification
  - Given x_q, find k nearest examples
  - Classify based on plurality vote w/ odd k
  - Low k can overfit, and high k underfit
  - Minkowski distance:

    $L_p(x_q, x_j) = (\sum_i |x_{j,i} - xq, i|^p)^{1/p}$
    - $p = 2$: Euclidean Distance
    - $p = 1$: Manhattan ""
  - Any other abstract distance function
  - Normalize if dimension scale changes
- **KNN average regression**
  - $h(x) = \sum_i y_i/k$, $y_i \in k$ nearest points.
  - larger k: smoother spikes; discontinuities
- **KNN linear regression**
  - Plot a line through k examples
  - Discontinuities still an issue
- **Locally weighted regression**
  - Smooth curve, avoid discontinuities
  - Use kernel function $K(dist(x_q, x_j))$
    - more weight on closer points
  - $w^* = \text{argmin}_w \sum_j K(d(x_q, x_j))(y_j - w \cdot x_j)^2$
  - $h(x_q) = w^* \cdot x_q$
  - Must solve for w^* for $\forall$ query points.
  - But only points overlapping kernel matter.

Ensemble Learning

- Vote among multiple hypotheses
- *Inductive Bias*: Mult. hyp. generalize better
- $\Pr\{h_k(x) \neq y_k\}$ indep for $\forall k$
- more complex h via linear combo of h_k.
- **Boosting/AdaBoost**
  - For each h_i (weak classifier)
    - correctly classified x: decrease weight
    - incorrectly classified: increase weight
  - Ultimately more emphasis on misclassified
  - Produce weighted average of all hypotheses
  - $f(x) = \sum_{t=1}^{k} \alpha_t h_t(x)$, a strong classifier
  - $\uparrow k \rightarrow$ near perfect training error
  - requires $h(i)$ has error rate $> 50\% + \epsilon$
  - Will overfit if weak learner always overfits
- **Random Forest**
  - *Analogy*: multiple interviewers
    - Randomly ask different questions
    - Vote on whether to hire candidate
  - Multiple decision trees vote on answer
  - Random sampling prevents overfitting
  - For each of k trees:
    - Randomly sample from a subset of X
    - "" "" subset of features
    - Learn tree from the sampled data

Neural Networks (NN)

- *Analogy*: brain neural connections and learning
- Input layer, $0 \leq$ hidden layers, output layer
- Domain knowledge helps establish structure
- Model *any* function with enough nodes/layers
- For each layer i w/ n_i inputs $\mathbf{a_i} \in \mathbb{R}^{n_i}$
  - Bias weight $b_i \in \mathbb{R}$
  - Input weights to next layer: $W_i \in \mathbb{R}^{n_i \times n_{i+1}}$
  - Activation function g
    - Apply to weighted sum of inputs + bias
    - $\mathbf{a_{i+1}} = g(W_i^\intercal a_i + b_i)$
- Perceptron (step) activation function
  - $g(z) = 1$if $z > 0, 0$otherwise
- Logistic/Sigmoid Function (Gaussian integrated):
  - $g(z) = \frac{1}{1+e^{-z}}$
- **Back Propagation** learning
  - Initiate input layer weights and bias
  - Observed error: $L_2(W) = |\mathbf{y} - h_W(\mathbf{x})|^2$
  - Update output weights via observed error
  - Use *gradient descent*
  - From output to earliest hidden layer:
    - Propagate Δ values to previous layer
    - Update weights between the two layers
  - (each hidden node resp. for some error)

Deep Learning

- Applications:
  - B/W image colorization
  - Sound generation from silent media
  - Machine translations
  - Object detection
  - Handwriting generation and recognition
  - [Eloquent] Text generation via recurrent NN
  - Game Playing (in a human-like way)
  - Speech recognition
  - Image object detection, classification, de-abstractization, sketch inversion
  - Natural Language Processing

Support Vector Machines (SVM)

- Maximum margin separator between examples
- Non-param., but needs frac. of ex. (support vectors)
- Expensive: use quadratic programming - $O(n^3)$
- Support vectors lie along decision boundary
- Kernels (kernel function)
  - notion of similarity between data
  - linearly-separable data:
    - linear kernel: $K(x, y) = x^\intercal y$
  - non-linearly separable data
    - polynomial: $K(x, y) = (x^\intercal y + c)^d$
    - project data to higher dimension
    - dividing hyperplane in $d + 1$ dimensions
      - d degrees of freedom
  - Radial-Basis: $K(x, y) = e^{-(\|x-y\|^2/(2\sigma^2))}$
  - Sigmoid: $K(x, y) = tanh(\alpha x^\intercal y + \theta)$
  - Kernels for other abstract similarity...
  - Requires some domain knowledge of data
- Regularization constant C and Soft-Margin SVM
  - $\downarrow C$: permit points to fall inside margin
- Multi-class classification: 1vs1 or 1vs$\forall$ strategies

Randomized Algorithms/Optimization

- Given i/p X, objective/fitness func. $f : X \mapsto \mathbb{R}$
  - find $x \in X$ such that $f(x) = \max_x f(x)$
- **Hill Climbing** (HC)
  - *Analogy*: Climb Everest in fog w/ amnesia
  - Start at arbitrary location in input space
  - Move to the highest-value neighbor
  - If neighbor > current, proceed to neighbor
  - Else return current. Possibly only *local max.*
- **Random Restart Hill Climbing** (RHC)
  - Alleviates local max somewhat
  - Restarts at random point a constant # of times.
- **Simulated Annealing** (SA, Metropolis Hastings)
  - *Analogy*: Rept'd heat/cool'g strengthens blade.
  - Allow bad moves, but with decreasing freq.

- T: some gradually decreasing temperature func.
- $\Delta E = Value(next) - Value(curr)$
- If $\Delta E \leq 0$ take bad move w/ prob. $e^{\Delta E/T}$
- As $T \to 0$, transitions *Random Walk* $\to$ *HC*.

- **Genetic Algorithms** (GA)
  - *Analogy*: Natural selection and mutation
  - *Apps*: Opt problems w/ approp. encoding
    - 8-Queens, circuit layout, job schedule
  - Select most fit pairs among population
  - Reproduce (cross-over) each pair
    - One-point crossover strategy
    - Uniform crossover strategy (random bits)
  - Mutate offspring with small probability
  - Replace least-fit individuals with new offspring
  - Repeat until convergence
- **MIMIC**
  - Model probability distribution instead of:
  - Population (non-parametric representation)
  - Convey structure
  - Generate samples from distribution $P^{\theta_t}(x)$
  - Set θ_{t+1} to n'th percentile
  - Retain only samples with $f(x) \geq \theta_{t+1}$
  - Estimate $P^{\theta_{t+1}}(x)$
  - Repeat until convergence
  - Technical details:
    - Estimate distribution via dependency tree
    - Use the *KL divergence* from info theory
    - Vastly fewer iterations than above algs
    - Each iteration much more costly

Bayesian

- Most prob. h given the data: $\text{argmax}_{h \in H} \Pr\{h|D\}$
- Bayes Rule: $\Pr\{h|D\} = \Pr\{D|h\} \Pr\{h\} / \Pr\{D\}$
- Maximum a posteriori (MAP):
  - $h_{MAP} = \text{argmax}_h \Pr\{D|h\} \Pr\{h\}$
  - disregard the normalizer $\Pr\{D\}$
- Maximum likelihood: $h_{ML} = \text{argmax}_h \Pr\{D|h\}$
  - assumes uniform $\Pr\{h\}$. Actual h irrelevant.
  - if Gaussian noise in data $\to \text{argmin}_h L_2$ error.
- Problems:
  - Requires domain knowledge of prior probabilities
  - Expensive to compute, being linear in $|H|$
- **Bayesian Optimal Classifier**
  - Most probable *classification* given the data
  - Combine weighted predictions of $\forall h_i$ given data
  - $V_{MAP} = \text{argmax}_v \sum_h \Pr\{v|h\} \Pr\{h|D\}$
- **Bayesian Belief Networks** (Bayes Nets)
  - Cond. indep. assumptions for X and Y given Z
    - $\Pr\{X|Y,Z\} = \Pr\{X|Z\}$
  - Conditional probabilities
  - Sample nodes in *Topological Sort*
  - *apps*: simulate/approximate a complex process
- **Naive Bayes**
  - assume data independence given parent.
  - $\to$ assume attr. independence given class.
    - No guarantee in the real world.
  - *apps*: classifying text documents
    - Attributes: words. Values: frequencies, OR
    - Attributes: word positions. Values: words.
  - $V_{NB} = \text{argmax}_V \Pr\{V\} \prod_i \Pr\{a_i|V\}$
  - Requires sufficiently large set of training data
  - Tractable: requires no search

Classification Metrics

Unsupervised Algorithms

Clustering

- **Single-Linkage Clustering**
  - Start with n points
  - Intercluster distance: closest two points in each
    - Alternatives: median, mean distance
  - Merge two closest clusters
  - Repeat $n-k$ times to produce k clusters
  - $O(n^3)$, but practically fast
- **K-Means Clustering** (in Euclidean Space)
  - 'Hard' clustering (special case of EM Clustering)
  - *Apps*: Image segmentation and compression
  - Pick k centers at random (or distributed)
  - Each center "claims" its closest points
    - Closest $\to$ minimizing L_2 error.
  - Recompute the centers (avg clustered points)
  - Repeat until convergence (to *local minimum*!)
  - $O(kn)$ per iteration, and $O(k^n)$ iterations.
- **K-Medoids Clustering**
  - Not Euclidean-Space $\to$ cannot use L_2 error.
  - Use abstract $\mathcal{V}(x_n, \mu_k)$ instead of $\|x_n - \mu_k\|^2$
  - Can't take average of clusters.
  - Can assign μ_k to one of cluster points
  - $O(kn + n_k^2)$ per iteration
- **Expectation-Maximization** (EM Clustering)
  - 'Soft' clustering
    - Point shared by mult. clusters prob'ly
  - Gaussian mixture model:
    - $f(x) = \sum_{i=1}^{k} \pi_i N_i(x|\mu_i, \sigma_i^2)$
    - π_i: mixing coefficient. $\sum_{i=1}^{k} \pi_i = 1$
  - Mixed Gaussian (μ, σ^2) for each cluster
  - Pick initial hidden vars: μ_k, covariances, and π_k.
  - $E-step$: prob. of component k explaining x
  - $M-step$: Re-estimate hidden vars
  - Iterate until log-likelihood reaches convergence
    - $\log \Pr\left\{\mathbf{X}|\mu, \boldsymbol{\sigma^2}, \pi\right\} = \sum_{n=1}^{N} \log\left\{\sum_{k=1}^{K} \pi_k N(x_n|\mu_k, \sigma_k^2)\right\}$
  - Local optima possible $\to$ random restart.

Dimensionality Reduction

- *Filtering*: Feature selection abstracted from learner
  - Fast, but ignore the learning problem
  - Ex: DT learning, information gain, variance
- *Wrapping*: Coupled with learner as one routine
  - Takes model bias into account, but SLOW.
  - Ex: Hill Climbing, Rand. Algorithms
- *Forward Search*:
  - Start with empty set of features.
  - Add most 'contributing' features until threshold.
- *Backward Search*:
  - Start with full set of features.
  - Remove most 'useless' features until threshold.
- **Principle Component Analysis** (PCA)
  - Mutually orthogonal and ordered transformed features
  - Gravitate towards 'average' of features
  - 'Best' dimension: highest *variance/eigenvalue*
  - This is the *First Principle Component*
    - Find via *SVD* in Linear Algebra
  - Find orthogonal (*Second Principle*) component
  - Idea: min. L_2 error moving from $N \to M$ dims
  - Select best components with non increasing var.
  - Eliminate dimensions beyond best M
- **Independent Component Analysis** (ICA)
  - Mutually independent transformed features
  - Gravitate towards 'abstracted' components
  - *Apps*: Cocktail party problem, mixed models
  - Min. *mutual information* for transf. features
    - Maximizing *kurtosis* of a dim. is one way.
  - $\to I(y_i; y_j) = 0$ (or minimum)
  - $I(X;Y)$ is max. (between *orig.* & transformed)
  - Idea: given observables, find indep. hidden var.
- **Random Component Analysis** (RCA)
  - Linear transformation in random directions
  - *Random* linear. combo of orig. dims still useful
  - For $N \to M$, M normally $\uparrow$ than PCA/ICA.
  - Fast. But requires many random trials.
- **Linear Discriminant Analysis** (LDA)
  - Finds projection that *discriminates* on label.

Classification Metrics

- Accuracy
- Precision
- Recall
- F_b

Reinforcement Learning

Markov Decision Processes (MDP)

- *Markov*: Only present matters. Stationary rules.
- $a \in Actions$, $s, s' \in States$
- *Model*: $T(s,a,s') \approx \Pr\{s'|s,a\}$
- *Reward*: $R(s), R(s,a), R(s,a,s')$ (typically 1st form)
- *Policy*: $\pi(s) \to a$.
  - π^*: optimal policy w/ max reward.
- *Infinite horizon*: navigate world forever, ∞ rewards.
- *Finite horizon*: discount γ to incentivise finish.
- $\pi^*(s) = \text{argmax}_a \sum_{s'} T(s,a,s')U(s')$
- **Bellman Equation**:
  - $U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$
  - $U(s)$: state utility, $R(s)$: state reward
  - n states $\to n$ equations, n, unknowns
  - non-linear due to *max* operation
- **Value Iteration** (VI)
  - Start w/ arbitrary utilities
  - Update based on neighbors

- $\hat{U}_{t+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')\hat{U}_t(s')$
  - Repeat until convergence
  - Then solve for π^*, straightforward
  - Lends itself for parallel computation
- **Policy Iteration** (PI)
  - Start with arbitrary π_0
  - Given π_t, calculate $U_t = U^{\pi_t}$ (follow policy)
    - $U_t(s) = R(s) + \gamma \sum_{s'} T(s,\pi_t(s),s')U_t(s')$
    - The action is fixed from the policy $\pi_t(s)$
    - n *linear* eq, n unknowns. $\rightarrow$ Linear Algebra.
  - Improve: $\pi_{t+1} = \text{argmax}_a \sum_{s'} T(s,a,s')U_t(s')$

Reinforcement Learning (model-free)

- No model (transition probabilities) or rewards
- Given transitions $< s, a, r, s' >$, learn policy
- **Q-Learning**
  - $Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$
  - Derive U and π from Q:
    - $U(s) = \max_a Q(s,a)$
    - $\pi(s) = \text{argmax}_a Q(s,a)$
  - Estimate from transitions (w/out R & T):
  - $\hat{Q}(s,a) \xleftarrow{\alpha_t} r + \gamma max_{a'}\hat{Q}(s',a')$
    - α_t: learning rate
    - $max_{a'}\hat{Q}(s',a')$: util. of next state
    - Notation: $v \xleftarrow{\alpha} x = v \leftarrow (1-\alpha)v + \alpha x$
  - $\hat{Q}$ starts anywhere (results will vary)
  - Following update rule above, $\hat{Q}(s,a) \rightarrow Q(s,a)$
  - Visiting s, a ∞ times, $s' \approx T(s,a,s'), r \approx R(s)$
  - Questions:
    - Initial $\hat{Q}$
    - α_t decay factor?
    - action choosing policy?
      - Take random action *sometimes*
      - Otherwise take best action
      - Mimics simulated annealing
  - *Exploration vs. Exploitation*
    - Exploration: cont. to learn Q
    - Exploitation: quicker maximize π^*

Game Theory

Zero-Sum Games

- **Minimax/Maximin**
  - Player A considers worse-case strategy by B
  - A chooses maximum minimum value by B
  - B chooses the minimum maximum value by A
  - Applies to *perfect* or *hidden* information games
- *Von Neumann*
  - 0-sum games of per. info: *minimax=maximin.*
  - $\exists$ optimal strategy for each player
- Optimal *Mixed Strategy* for 2X2 game
  - Two players, A & B, strategies A_1, A_2, B_1, B_2
  - Mixed strategy gains: $\mathbf{m_{AB}} : m_{11}, m_{21}, m_{12}, m_{22}$
  - A uses strategy 1 w/ probability p
  - eg_i: Expected gain for A given B uses strategy i
  - $eg_1 = m_{11}p + m_{21}(1-p)$
  - $eg_2 = m_{12}p + m_{22}(1-p)$
  - $p^* = \max_p \{\min\{eg_1, eg_2\}\}$
    - Either $p = 0, p = 1$, or where both equal
- Optimal *Mixed Strategy* for nXm game
  - $\mathbf{P_A} = \{p_1, p_2, ..., p_n\}$
  - $eg_j = \sum_{i=1}^{n} m_{ij}p_i$ for all $j \in 1, ..., m$
  - n unknowns, m eq. $\rightarrow$ Linear Programming
    - Choose $\mathbf{P_A}$ to maximize $\min\{eg_1, ..., eg_m\}$
    - such that $\sum p_i = 1, 0 \leq p_i \leq 1.\forall i$

Non-Zero-Sum Games

- Non-deterinistic, non-cooperative, hidden info
- Typical example: Prisoner's Dilemma
- n strategy spaces $\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_n$
- n payoff functions $u_1, ..., u_n$ st $u_i : \mathbf{S}_1 \times \cdots \times \mathbf{S}_n \rightarrow \mathbb{R}$
- What's the optimal mixed strategy?
- **Nash Equilibrium** (NE)
  - $s_1^*, ..., s_n^* \in S_1 \times \cdots \times S_n$ are a NE iff
    - $\forall i. s_i^* = \text{argmax}_{s_i} u_i(s_1^*, ..., s_i, ..., s_n^*)$
    - ($\neg\exists$ reason for any one player to switch)
  - Works for pure & mixed strategies
  - Assumes best action for self (regardless of others)
  - Can contain multiple Nash Equilibria
  - An *implausible* threat hinders own utility
  - n *repeated games* $\rightarrow n$ repeated NE
    - assuming implausible threats

Bayesian Games

- Action spaces: $\mathbf{A_1}, ..., \mathbf{A_n}$
- Type spaces: $\mathbf{T_1}, ..., \mathbf{T_n}$
- Beliefs: $\mathbf{P_1}, ..., \mathbf{P_n}$
- $P_{-i}(t_{-i}|t_i)$ = PDF of others' types given own
- Payoff functions: $u_1, ..., u_n$
- $u_i(a_1, ..., a_n, t_i)$ - payout to player i w/ type i
  - player j chooses action a_j for $\forall j$
- All players know their own $\mathbf{A_i}, \mathbf{T_i}, \mathbf{P_i}, u_i$
- Strategy $S_i : \mathbf{T_i} \rightarrow \mathbf{A_i}$
- **Bayesian Nash Equilibrium** (BNE)
  - $s_1^*, ..., s_n^* \in S_1 \times \cdots \times S_n$ are a BNE iff $\forall i$ and $\forall t_i \in \mathbf{T_i}$. $s_i^*(t_i) = \text{argmax}_{a_i \in \mathbf{A_i}}$ $\left\{\sum_{t_{-i} \in \mathbf{T_{-i}}} u_i(s_1^*(t_1), ..., a_i, ..., s_n^*(t_n)) \times P_i(t_{-i}|t_i)\right\}$

Repeated games w/ uncertain end

- # rounds left is uncertain
- Pr$\{playagain\} = \gamma$, Pr$\{gameover\} = 1 - \gamma$
- E$[\#rounds] = 1/(1-\gamma)$
- **Tit for Tat** strategy
  - Cooperate 1st round, copy opponent move after
- **Grim Trigger** strategy
  - Cooperate while opponent cooperates.
  - Once line is crossed, forever defect
- **Pavlov** strategy
  - Cooperate if agree, defect if disagree.
  -
- *Subgame perfect* (SP)
  - Str. is always best response indep. of history
  - $\neg$SP if $\exists$ history of moves st. str. implausible
  - $\neg SP \iff \exists$ pair of str's not leading back to mutual cooperate
- *Mini-Max Profile* (For zero-sum game)
  - Min guaranteed payoffs for $\forall$ players on defense
  - Applicable in pure or mixed strategies

Stochastic Games & Multiagent RL

- *Analogy*: MDP-RL::Stochastic game-Multiagent RL
- More general than MDPs or other previous models
- $\mathbf{S}$: states
- $\mathbf{A}_i$: actions for player i, $a, b, a \in \mathbf{A}_1, b \in \mathbf{A}_2$
- $\mathbf{T}$: transition probabilities $T(s,(a,b),s')$
- R_i: rewards for payer i, $R_1(s,(a,b)), R_2(s,(a,b))$
- γ: discount
- Impose restrictions to produce other models:
  - $R_1 = -R_2 \rightarrow$ 0-sum stochastic game
  - $T(s,(a,b),s') = T(s,(a,b'),s')$, $R_2(s,(a,b)) = 0, R_1(s,(a,b)) = R_1(s,(a,b'))$ $\forall b' \rightarrow$ MDP, makes second player irrelevant
  - $|\mathbf{S}| = 1 \rightarrow$ repeated games
- **Zero-Sum Stochastic Games**
  - $Q_i^*(s,(a,b)) = R_i(s,(a,b)) + \gamma \sum_{s'} T(s,(a,b),s') minimax_{a',b'} Q_i^*(s',(a',b'))$
  - *Q-Learning* for transition $< s, (a,b), (r_1, r_2), s' >$
    - $Q_i(s,(a,b)) \xleftarrow{\alpha} r_i + \gamma minimax_{a',b'} Q_i(s',(a',b'))$
    - Value Iteration works
    - Minimax-Q converges
    - Unique solution to Q^*
    - Policies can be computed independently
    - Update efficient (polytime)
    - Q functions sufficient to specify policy
- **General-Sum Stochastic Games (Nash-Q)**
  - Same as Q_i^*/Q-Update above, but use Nash Equilibrium instead of minimax/maximin
  - Value Iteration <u>doesn't</u> work
  - Nash-Q <u>doesn't</u> work
  - No unique solution
  - Policies <u>can't</u> be computed independently
  - Update <u>not</u> efficient
  - Q functions <u>not</u> sufficient to specify policy

Vitaly Parnas, 2018