1 Maschine Learning

Algorithm learns class of tasks, measured by loss function, from experience.

supervised learning learn $h: \Delta^* \to \Sigma^*, h = t$; example: $(x,y) \in \Delta^* \times \Sigma^*$, $t(x) = y$.

unsupervised learning learn $h: \Delta^* \to \Sigma^*, \ker(h) = \ker(t)$; example: $x \in \Delta^*$.

reinforcement learning learn strategy based on feedback from environment.

2 Supervised Learning

- model function $t: \mathcal{M} \to \mathcal{R}$
- $supp(t) = \{m \in \mathcal{M} \mid t(m) \neq 0\}$
- $\bar{m} \in supp(t) \Leftrightarrow t(\bar{m}) = 1$

Hypothesis of A: potential result of A

Hypothesis space $\mathcal{H}_A$ of A: set of all hypotheses

h fits D if $h(x_i) = y_i$ for all $(x_i, y_i) \in D$

Version space $\mathcal{V}_A(D)$ of A: all hypotheses that fit D

Inductive bias of A: set of assumptions that A uses to predict outputs of unseen data

2.1 Conjunctive Clause

$\theta = (\theta_1, ..., \theta_k), \theta_i \in M_i \cup \{\star, \perp\}$

- $\theta_\perp = (\perp, ..., \perp)$ most specific
- $\theta_\star = (\star, ..., \star)$ most general
- $supp(h_{\theta_\perp}) = \emptyset, supp(h_{\theta_\star}) = \mathcal{M}$
- $h_{\theta_\perp} = h_{(\theta_1, ..., \perp, ..., \theta_k) = ...}$

induced hypothesis $h_\theta(m_1, ..., m_k) = 1$ if $\forall i: \theta_i \in \{m_i, \star\}$ else 0

$h \preceq h'$ if $supp(h) \subseteq supp(h')$. h is more specfic (less general) than h'

Find-S Algorithm finds most specific conjunctive clause that fits D

1. Start with $\theta_\perp = (\perp, ..., \perp)$
2. iterate over POSITIVE examples
3. min-generalize θ to fit example
4. $\perp \to a, a \to \star$

- maximal general hypothsis:
  1. start at $\theta_\star = (\star, ..., \star)$
  2. exclude every negative example
  3. $(\star, ...) \to \{(b, ...), (c, ...)...\}$
- If $V_A(D) \neq \emptyset$, Find-S finds $h \in V_A(D)$

disjunctive normal form $\Theta = \{\theta_1, ..., \theta_m\}$

'finite set of conjunctive clauses'

induced hypothesis $h_\Theta(\bar{m}) = 1$ if $\exists \theta \in \Theta: h_\theta(\bar{m}) = 1$ else 0

- $supp(\Theta) = \bigcup_{\theta \in \Theta} supp(h_\theta)$
- can represent all boolean functions

Boundary sets of version space

maximally general hypotheses $V_A^\top(D) = \{h \in V_A(D) \mid \nexists h' \in V_A(D): h \prec h'\}$

maximally specific hypotheses $V_A^\perp(D) = \{h \in V_A(D) \mid \nexists h' \in V_A(D): h' \preceq h\}$

-$h \in V_A^\top$ maximal, weil: $\forall x \in M \setminus supp(h): supp(h) \cup \{x\} \notin supp(V_A(D))$

Theorem: $V_A(D) = \{h \in H_A \mid \exists h_\top \in V_A^\top(D), \exists h_\perp \in V_A^\perp(D): h_\perp \preceq h \preceq h_\top\}$

-> $V_A(D)$ det. by $V_A^\top(D)$ and $V_A^\perp(D)$

- only 1 lower bound (in $V_A^\perp(D)$), potentially multiple upper bounds (in $V_A^\top(D)$)

Candidate Elimination Algorithm

Output: DNF for $V_A^\top(D)$ and $V_A^\perp(D)$

1. $S_\perp = \{\theta_\perp\}, S_\top = \{\theta_\star\}$
2. for $1 \leq i \leq n: y_i = 1$ (pos. xmpls)
   1. keep only fitting h from $S_\top$
   2. $\forall \theta \in S_\perp: h_\theta(x_i) = 0$
      - remove θ, add all min generalizations θ' of θ that fit x_i to $S_\perp$
   3. keep only most specific h in $S_\perp$
3. for $1 \leq i \leq n: y_i = 0$ (neg. xmpls)
   1. keep only fitting h from $S_\perp$
   2. $\forall \theta \in S_\top: h_\theta(x_i) = 1$
      - remove θ, add all min specializations θ' of θ that fit x_i to $S_\top$, for which a more specific $\theta_\perp \in S_\perp$ exists!
   3. keep only most general h in $S_\top$

- $V_A^\top = \{h_\theta \mid \theta \in S_\top\}, V_A^\perp = \{h_\theta \mid \theta \in S_\perp\}$
- Concept indentified if: $S_\perp = S_\top$ and $|S_\top| = 1$. $V_A(D) = \emptyset$ if $S_\perp = \emptyset \vee S_\top = \emptyset$

2.2 Decision Trees

Splitting $\Pi = \{M - 1, ..., M_p\}$ is finite partition of (sub)feature Space $\mathcal{M}'$

- induces splitting of $\{1, ..., n\}$ into $I_{D'}(M_1), ..., I_{D'}(M_p)$ (sets of indices)
- monothetic splits: based on 1 feature
- simple split: monothetic, into all realizations $M = \{\bar{m} \in M \mid m_1 = a(, b...)\}$
- binary split: monothetic, into 2 sets $M = \{\bar{m} \in M \mid m_1 \in A\} \cup \{\bar{m} \in M \mid m_1 \notin A\}$
- induced hypothesis $h_T(\bar{m}) = T(v)$, where v is unique leaf s.t. $\bar{m} \in M_v$
- simple decision trees can represent all hypotheses

Decision Tree Quality Measures

- Number of leaves
- Height (max number of constraints to check)
- External path length (sum of all path lengths from root to leaf)
- Weighted external path length (sum of all path lengths from root to leaf, weighted by number of examples classified in that leaf)

Theorem: Given D and bound b, its NP hard to decide existance of decision tree T s.t. h_T fits D and T has ext. p.l. $\leq b$

- Majority Class $Maj_D(M')$ maj. $r \in R$

- Number of Misclassifications: $Err_D(M', r)$ in feature subspace M' with majority class r
- $Err_D(T)$: sum up all $Err_D(M_v, T(v))$

Pure Node v if $Err_D(M_v, T(v)) = 0$

- class distribution $p_D^{M'}(r)$: $p(r)$ in M'
- **Impurity Function** $\iota: [0,1]^R \to \mathbb{R}$ if
  - $\iota(p)$ is minimal $\forall p: p(r) = 1$
  - ι symmetric in classes
  - ι is maximal for uniform distr.

gets propability distribution as input

1. $\bar{\iota}(p) = 1 - \max_{r \in R} p(r)$

2Entropy $H(p) = -\sum_{r \in R} p(r) \log_2 p(r)$

3.Gini Impurity $G(p) = 1 - \sum_{r \in R} p(r)^2$

- Impurity of M' is $\iota_D(M') = \iota(p_D^{M'})$

Impurity Reduction of splitting $\Pi = \{M_1', ..., M_p'\}$ of M' is:

$\iota_D(\Pi) = \iota_D(M') - \sum_{i=1}^p \frac{|I_D(M_i')|}{|I_D(M')|} \iota_D(M_i')$

Tree Construction for $M' \subseteq M$

1. if no elements in M': new leave v: $T(v) = Maj_D(M)$

2. if $\iota(M') \leq \epsilon$: new leaf v: $T(v) = Maj_D(M')$

3. else: select split Π of M' with maximal impurity reduction

- strict imp. fct.: concave at every point
- $\iota_D(\Pi) \geq 0 \ \forall \ \Pi$ and strict imp. fct. ι

ID3 simple D.T., monothetic simple splits, impurity function: entropy. inductive bias: local optimization (greedy)

CART D.T., binary splits, impurity function: Gini impurity

- true loss of $h \in H_A$: misclassifications: $l^*(h) = \sum_{\bar{m} \in M}(1 - \delta_{h(\bar{m}), t(\bar{m})})$
- h **overfits** D if $\exists h' \in H_A$: $l(h, D) < l(h', D)$ and $l^*(h) > l^*(h')$

when: training data: noisy, small, biased

- Training Data: optimize loss here
- Validation: optimize hyperparameters
- Test Data: final estimation (true loss)
- h **overfits** (D, D_V) if $\exists h' \in H_A$: $l(h, D) < l(h', D) \wedge l(h', D_V) < l(h, D_V)$

true loss of h estimated by $l(h, D_V)$

Countermeasures to overfitting:

- increase data quality/quantity
- early stopping (no more splits) thrhld large -> omits useful splits thrhld small -> Large Tree
- regularization (penalize model complexity in training process)

Pruning: turn inner node v into leave with label $Maj(M_v)$

D.T. pruning Algorithm

1. given fully trained D.T.

2. prune every inner node als long as pruning doesnt increase validation loss: $l(h_T', D_V) \leq l(h_T, D_V)$

2.3 Linear Regression

$t: M \to R, M \subseteq \mathbb{R}^k, R \subseteq \mathbb{R}$

- find $w = (w_0, ..., w_k) \in \mathbb{R}^{k+1}$ s.t.: $h_w(x_1, ..., x_k) = w_0 x_0 + \sum_{i=1}^k w_i x_i$ approximates $t \Leftrightarrow w$ minimizes $l(h_w, D)$
- note: $x_0 = 1$ always! (w_0 is bias)

Analytical Solution

1. Partial derivatives: $\frac{\partial l(h_w, D)}{\partial w_i}$
2. Set to 0, put in values from D
3. Solve LGS with Gauss for w_i

SGD (Iterative Solution)

1. initialize w randomly
2. choose random $1 \leq i \leq n, T++$
3. $\delta = y_i - h_w(x_i)$ (residual)
4. $\Delta w = \delta \cdot x_i$ (derivatives)
5. $w = w + \eta \cdot \Delta w$ (parameters)
6. If $\neg$converged $\to$2, else return w

- Pros: simple, robust to noisy data, representation independent
- Cons: stability, convergence problems, sensitive to learning rate η

BGD (accumulate derivatives $\forall i$)

1. initialize w randomly
2. For each $1 \leq i \leq n, T++$:
   2.1. $\delta = y_i - h_w(x_i)$
   2.2. $\Delta w = \Delta w + \delta \cdot x_i$
3. $w = w + \eta \cdot \Delta w$
4. If $\neg$converged $\to$2, else return w

- sequence of examples (batch) are processed together, before updating w

IGD

1. initialize w randomly
2. For each $1 \leq i \leq n, T++$:
   2.1. $\delta = y_i - h_w(x_i)$
   2.2. $\Delta w = \delta \cdot x_i$
   2.3. $w = w + \eta \cdot \Delta w$
3. If $\neg$converged $\to$2, else return w

Property	SGD stochastic	IGD iterative	BGD batch	MBGD mini-batch
Batch size	1	1	n	varies
Batch selection	random	sequential	sequential	sequential
Parallelization	difficult	difficult	trivial	trivial
Space requirement	low	low	high	varies
Stuck local minimum	no	no	yes	varies
Convergence speed	slow	slow	fast	varies

Polynomial Regression

Approach: 1. prepare nonlinear combinations of features as features (curse of dimensionality: max k features $5k \leq n$)

2. then perform linear regression on **expanded** feature space with SGD, BGD, IGD or analytical approach.

3. Project solution back to input (feature) space

- keep original features
- for m_i^3 also include m_i^2
- increase complexity -> increase risk of overfitting

Regularization

'penalize model complexity in training process', optimize for $l'(w, D)$: $l'(w, D) = l(h_w, D) + \frac{\lambda}{k} \cdot r(w)$

- Lasso Regression: $r(w) = \sum_{i=1}^k |w_i|$
- Ridge Regression: $r(w) = \sum_{i=1}^k w_i^2$
- λ big -> more regularization -> less complex model
- k: num of features (excluding bias w_0)

Develop (S,B,I)GD for **specific loss function** $l(x)$:

1. get 1st derivative $l'(x)$ of loss: for 1 Data example: $n = 1$, leave out $\sum$

2. find $-\delta = h_w(x_i) - y_i$ in 1st derivative

3. replace line 4 (derivatives) with: $\Delta w = l'(x)$ but substitute δ (!- $-$ 1!)

Logistic Regression (Classification)

'find optimal hyperplane w' by optimization, to get h_w, to get classifier: $h_w^c(z) = 1$ if $h_w(z) \geq \frac{1}{2}$, 0 otherwise

- 'discriminative' classifier
- only reasonable for binary $R = \{0, 1\}$ ($|R| > 2$ induces order bias)
- Training: optimizes w for $l(h_w, D)$
- Prediction: performed by h_w^c (different)
- Discriminating Hyperplane 1 Dimension less than w (Plane -> Line -> Point)

> logistic function: $h_w^\sigma(z) = \sigma(h_w(z))$

> logistic classifier: $h_w^{c,\sigma}(z) = 1$ if $h_w^\sigma(z) \geq \frac{1}{2}$, 0 otherwise

- 'generative' classifier
- $h_w^\sigma(z)$ gives prop, that z is class 1

> MLE Maximum Likelihood Estimator given H_A for D is: $\hat{h} = \arg\max_{h \in H_A} P[D;h]$

2.4 Support Vector Machines

'learn optimal discriminating Hyperplane with maximal margin directly'
$H(w) = \{(z_1, ..., z_k) \in \mathbb{R}^k | w_0 + w_1 x_1 + ... + w_k x_k = 0\}$
- Normal Representation $w = (w_0, ..., w_k)$ is normal to discriminating Hyperplane $H(w)$
- H_1 closest x_i with $y_i = 1$ $(wz^T = 1)$
- H_0 closest x_i with $y_i = 0$ $(wz^T = -1)$
- Hyperplanes H_1, H_0 parallel to $H(w)$

Margin distance$(H_1, H_0) = \frac{2}{||\vec{w}||}$
- Hinge Loss: only falsly classified data causes loss

Hard Margin SVM: no misclassifications/boundary violations
- $\hat{w} = arg\min_w \frac{1}{2}\vec{w}\vec{w}^T$, $l_h(h_w, D) = 0$

Soft Margin SVM: λ trades margin size against boundary violations
λ small: larger margin, more violations
λ big: smaller margin, less violations
- $\hat{w} = arg\min_w \frac{1}{2}\vec{w}\vec{w}^T + \lambda l_h(h_w, D)$

Kernel Trick: Kernels permits nonlinear seperation in input space $\mathbb{R}^k$ (through linear seperation in $\mathbb{R}^{k+d}$)
- with suitable Kernel: no actual computations in $\mathbb{R}^{k+d}$
-> no additional effort for non linear classification (linear classifier for free)

D Linearly Seperable if:
$\exists w_0, w_1, ..., w_k : y_i' \cdot (w_0 + w_1 x_1 + ... + w_k x_k) > 0$
where: $y_i' = 1$ if $(y_i = 1)$, -1 if $(y_i = 0)$
1. $\forall i$ where $y_i = 1 : h(x_i) > 0$
2. $\forall i$ where $y_i = 0 : h(x_i) < 0$

Proof by finding w, then transform to discriminating hyperplane (eg point x):
- $w_0 + w_1 x_1 = 0 \Rightarrow x = -\frac{w_0}{w_1}$

Disprove by finding contradiction in System of inequations.

2.5 Neural Networks

Perception Hypothesis
$h_w^H(z) = H(\sum_{i=0}^k w_i z_i) = 1$ if $wz^T \geq 0$

Perceptron Training
1. initialize w randomly, $T = 0$
2. Select random $1 \leq i \leq n$
3. $\delta = y_i - h_w^H(x_i)$
4. $\Delta w = \delta \cdot x_i$
5. $w = w + \eta \cdot \Delta w$
6. If $l(h_w^H, D) \neq 0$ to 2, else return w

Rosenblatt: If D linearly seperable: PT terminates after finitely many corrections

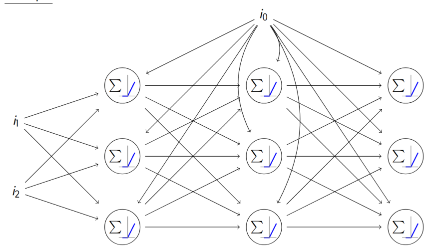
Property	Gradient descent	PT algorithm
Loss function Discriminator	ℓ_2 or ℓ_σ hyperplane	0-1 loss $\ell_{0/1}$ hyperplane
Data inseparable Perfect separation	robust potentially	no termination guaranteed
Parameter updates on Correction size	surrogate error scaled by steepness	class error fixed

Nonlinear seperation possible through network of perceptrons

Activation	Function	Symbol	Algorithm
Linear	$\mathrm{id}(z) = z$	Σ	Linear regression
ReLU	$\mathrm{ReLU}(z) = \max(0, z)$	Σ	–
Leaky ReLU	$r_\alpha(z) = \max(0,z) + \alpha \cdot \min(0,z)$	Σ	–
Heaviside	$H(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$	Σ	PT algorithm
Sigmoid	$\sigma(z) = \frac{1}{1+e^{-z}}$	Σ	Logistic regression

Network $N = (V, E, wt)$, $wt : E \to \mathbb{R}$
- State: $s : V \to \mathbb{R}$
$s'(v') = a(\sum_{v \in V} s(v) \cdot wt(v, v'))$
a(weighted sum of previous states)
- OR: $N = W \in \mathbb{R}^{(V \cup I) \times V}$ (adj matrix)
- softmax $\sigma : \mathbb{R}^V \to \mathbb{R}^V$ outputs distribution over inputs
- FeedForward N: no loops, no cycles, every node reachable from 1 input node

Example



$w_0 * i_0 = w_0$ is bias, noted above node
Characteristics: 1. input, 2. internal nodes, 3. layers, 4. type (FFN/RNN?)

Induced Regression Hypothesis
$h_W^v(z) : \mathbb{R}' \to \mathbb{R}, h_W^v(z) = s_v^{(m)}$
Forward Pass with fixed output node

Neural Network Training
compute gradients gradients
> Gradients of Nodes $\frac{\partial L}{\partial r_v} =$
1. $(s_v - y) \cdot H(r_v)$ if v designated o node
2. 0 if v other o node
3. $H(r_v) \cdot \sum_{v' \in V} W_{vv'} \cdot \frac{\partial L}{\partial r_{v'}}$ otherwise

> Gradients of Edges $\frac{\partial l}{\partial W_{vv'}}$: 'State of predecessor $\cdot$ gradient of successor'

FF NN Training with SGD
1. initialize W randomly
2. choose random $1 \leq i \leq n, T++$
3. compute states s_v (forward pass)
4. compute gradients ΔL for (x_i, y_i) (backward pass)
5. $W = W - \eta \cdot \Delta L$ (update weights)
6. If $\neg$converged $\to$2, else return W

- RNNs good for sequential data, gives 'sequence regression hypothesis'

m-unroll of RNN: FNN with m layers.
backpropagation through time: unroll RNN, compute gradients for each layer, then average gradients for each parameter over all layers

loss functions (and derivatives)

- $l(h,D) = \sum_{i=1}^{n}(1-\delta_{y_i,h(x_i)})$
- $\delta_{ij} = 1$ if $i=j, 0$ otherwise.
- $l_2(h,D) = \frac{1}{2n}\sum_{i=1}^{n}(h(x_i)-y_i)^2$ (Mean Squared Error)
  $\frac{\partial l(h,D)}{\partial w_p} = \frac{1}{n}\sum_{i=1}^{n}(h(x_i)-y_i)x_p = \frac{1}{n}\sum_{i=1}^{n}(w_0+w_1x_{i1}+...+w_px_{ip}-y_i)x_p$
- $l'(w,D) = \frac{1}{2n}\sum_{i=1}^{n}(h(x_i)-y_i)^2 + \frac{\lambda}{k}\sum_{i=1}^{k}w_i^2$(Ridge Regression)
  $\frac{\partial l'(w,D)}{\partial w_p} = \frac{1}{n}\sum_{i=1}^{n}(h(x_i)-y_i)x_p + 2\lambda w_p,\ w_p \in \{0, w_1, ...w_k\}$ WARUM NICHT: ... + $\frac{2\lambda}{k}w_p$?!?!?!??!?!?!
- $\ell_\sigma(h,D) = -\frac{1}{n}\sum_{i=1}^{n}\Big(y_i\cdot\log\big(h(x_i)\big)+(1-y_i)\cdot\log\big(1-h(x_i)\big)\Big)$(Logistic Loss)
  $\frac{\partial \ell_\sigma(h_w^\sigma,D)}{\partial w_p} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i - h_w^\sigma(x_i)\right)\cdot x_{ip}$
- $l_h(h,D) = \frac{1}{n}\sum_{i=1}^{n}\max(0, 1-(2y_i-1)h(x_i))$ (Hinge Loss)

Ableitungsregeln

- Produktregel: $(f\cdot g)(x)' = f'(x)\cdot g(x) + f(x)\cdot g'(x)$
- Quotientenregel: $\left(\frac{f}{g}\right)'(x) = \frac{f'(x)g(x)-f(x)g'(x)}{g^2(x)}$
- Kettenregel: $(f\circ g)'(x) = (f'\circ g)(x)\cdot g'(x) = f'(g(x))\cdot g'(x)$
  $(f\circ g\circ h)'(x) = f'(g(h(x)))\cdot g'(h(x))\cdot h'(x)$
- $\left(\frac{1}{g}\right)'(x) = -\frac{g'(x)}{g^2(x)}$
- $\left(\frac{1}{x^n}\right)' = -nx^{-n-1}$
- $\log_a'(x) = \frac{1}{x\ln(a)}$
- $\ln_e(x)' = \frac{1}{x}$
- |x|' = $\frac{x}{|x|} = sgn(x)$ for $x \neq 0$