Machine Learning

(What will happen)

Three kinds of programs: (i) **Supervised**: Given input/output examples, finds mapping (**Predictive**) (ii) **Unsupervised**: Given data finds a representation, (**Descriptive**, What happened) (iii) **Reinforcement**: Translate state to action to maximize reward.

ML-Algorithm = Representation + Loss Function + Optimizer, h: hypotheses

Empirical Risk $L_S(h) \overset{def}{=} \frac{1}{m}\sum_{i=1}^{m} l_{0-1}(h, \vec{z}_i)$ (Average loss computed over each example)

$l_{0-1}(h,(\vec{x},y)) \overset{def}{=} \begin{cases} 0, & h(x)=y \\ 1, & h(x)\neq y \end{cases}$. **Empirical Risk Minimization**: $h_S \overset{def}{=} ERM_H(S) \in \underset{h\in H}{\operatorname{argmin}}\, L_S(h)$

Assume $S=\{X_i\}^m \sim D^m$ i.e. $S\sim D^m$; $L_D(h) \overset{def}{=} \underset{z\sim D}{\mathbb{E}}[l(h,\vec{z})]$ (**Test Data**). Fundamental Challenge in ML: $L_S(h) \neq L_D(h)$. **Overfitting**: $h(x) = \begin{cases} y_i, & \exists i \in \varepsilon \\ -1, & \text{otherwise} \end{cases}$. Hypothesis class $\mathcal{H}$ cannot include all hypotheses we must choose $\mathcal{H}^* \subset \mathcal{H}$.

Empirical Risk Minimization: ERM: Pick hypothesis that minimizes the loss on a set of training data (Naively ERM can lead to overfitting). **Affine Function**: $h_{w,b}(x) = \langle w, x\rangle + b$, $x\in\mathbb{R}^d$, $b\in\mathbb{R}$

$h_w(x) = \text{sign}(\langle w, x\rangle)$, we can use an affine function to define a hypothesis called a halfspace $\mathcal{X}=\mathbb{R}^d$, $y\in\{-1,1\}$, In a d-dimensional attribute space, the decision boundary is the $(d-1)$-dimensional hyperplane where $\langle w, x\rangle = 0$, w is a vector normal to that hyperplane (pointing towards positive side)

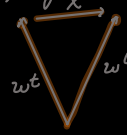
Linear Separation: What types of data can we separate perfectly with a halfspace i.e. when $\exists \vec{w}$ s.t. $L_D(h_w)=0$ $L_S(h_w)=0$ $\forall S\sim D$. Let $f:\mathcal{X}\to Y$ be the true-labelling function i.e. $D(y=f(x)\,|\,x)=1$, let $D(x)$ be uniform. Suppose $\mathcal{X}=\{0,1\}^d$, $Y=\{1,-1\}$ and $f(x) = \begin{cases} 1, & \text{if } \sum_d^d x_i \geq 1 \\ -1, & \text{otherwise} \end{cases}$

What's linearly separable: $f^A(x) = \neg x_1$, $f^B(x) = \bigwedge_{i=1}^{d} x_i$, $f^C(x) = x_1 \text{ xor } x_2$ d) $f^D(x) = \begin{cases} 1, & \sum_1^d x_i = 1 \\ -1, & \text{otherwise} \end{cases}$

Perceptron Algorithm: Iterative Algorithm, Initialize weights to be all zeroes

$w^{(1)} = (0,0,\dots,0)$
for $t=1,2,\dots$:
  If $(\exists i \text{ s.t. } y_i\langle w, x_i\rangle \leq 0)$
  else $w^{(t+1)} = w^{(t)} + y_i x_i$
  return $w^{(t)}$

Halfspace is correct if $y\langle w, x\rangle > 0$. Suppose for some training example that $y\langle w, x\rangle \leq 0$

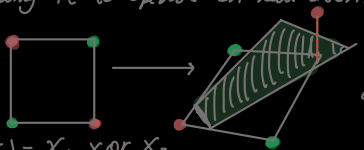


$y\langle w^{(t+1)}, x\rangle = y\langle w^{(t)} + y\langle x), x\rangle$
$= y\langle w^{(t)} + x\rangle + \|x\|_2^2$

Theorem: Assume $(x_1,y_1),\dots,(x_m,y_m)$ is separable, let $B = \min\{\|w\| : \forall i\in[m], y_i\langle w, x_i\rangle \geq 1\}$, $R = \max_i \|x_i\|$. Then, the algorithm stops after $(RB)^2$ at most iterations.

More dimensions: We could make a 2-d data set into a 3-d linearly separable data set through feature engineering i.e. expanding $\mathcal{H}$ to operate on new attributes $x' = \psi(x)$ where $\psi(x) = (x_1, x_2, x_1\cdot x_2)$

i.e.



$f^C(x) = x_1 \text{ xor } x_2$

We can place $d+1$ points in $\mathbb{R}^d$ and linearly separate any labels assigned to them Ex: $0, e_1=(1,0,0,\dots), e_2=(0,1,0,\dots), \dots$ then $\forall y_1, y_2, \dots$, set b to y_1 and w to $y_2-y_1, y_3-y_1, \dots$ (Shattering).

Linear and Polynomial regression: $\mathcal{X}=\mathbb{R}^d$, $Y=\mathbb{R}$, $h_{w,b}(x) = \langle w, x\rangle + b$

(MSE): $L_S(h_w) = \frac{1}{m}\sum_{i=1}^{m}(h_w(x_i)-y_i)^2$, Another option is $L_S(h_w) = \frac{1}{m}\sum_{i=1}^{m}|h_w(x_i)-y_i|$. Rewrite:

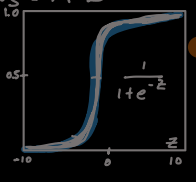
Least Squares: $\underset{w}{\operatorname{argmin}}\, L_S(h_w) = \underset{w}{\operatorname{argmin}}\, \frac{1}{m}\sum_{i=1}^{m}(\langle w, x_i\rangle - y_i)^2$, $\frac{2}{m}\sum_{i=1}^{m}(\langle w, x_i\rangle - y_i)x_i = 0$, $Aw = b$ where $A = \left(\sum_{i=1}^{m} x_i x_i^T\right)^{-1}$ $b = \sum_{i=1}^{m} y_i x_i$, If A not invertible: do eigenvalue decomposition $A = VDV^T$. Now define D^+ where D^+_{ij} is D_{ij} if D_{ij} is nonzero and 0 otherwise. Define $A^+ = VD^+V^T$ (**Moore-Penrose Pseudo-Inverse**) then $w_S = A^+ b$

If A invertible: $w_S = A^{-1}b$ (A invertible iff $\{x_1,\dots,x_d\}$ span $\mathbb{R}^d$)

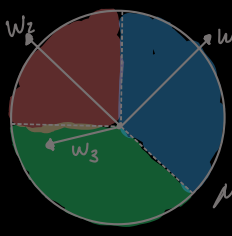
Logistic Regression: $\mathcal{X}=\mathbb{R}^d$, $y=\{1,-1\}$, $h_{\vec{w}}(x) = \frac{1}{1+e^{-\langle w, x\rangle}}$ continuous value in $[0,1]$ $h:\mathcal{X}\to[0,1]$

Multiclass L.R.: $\mathcal{X}=\mathbb{R}^d$, $y=\{1,\dots,k\}$, $h_w(x)_j = \frac{e^{\langle w_j, x\rangle}}{\sum_1^k e^{\langle w_s, x\rangle}}$ (**Softmax**), $\sum_{j=1}^{k} h_w(x)_j = 1$

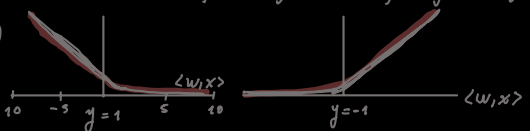
$h:\mathcal{X}\to[0,1]^k$, $w\in\mathbb{R}^{k\times d}$; $\begin{pmatrix} h_w(x)_1 \\ \vdots \\ h_w(x)_k \end{pmatrix}$ ← $\begin{pmatrix} \langle w_1, x\rangle \\ \vdots \\ \langle w_k, x\rangle \end{pmatrix}$ of



Decision Boundary $\langle w, x\rangle = 0$



(i) Larger magnitude push boundary toward 90° (ii) Boundary can't be more than 90° from weight vector if neighbor weight < 90°



Log Loss: Binary log-loss: $l(h_w,(x,y)) = \log(1+\exp(-y\langle w,x\rangle))$

Multiclass log loss: $l(h_w,(x,y)) = -\sum_{j=1}^{k} \mathbb{1}_{[y=j]} \log h_w(x)_j$

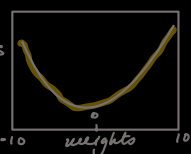
$L_S(h_w) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} \mathbb{1}_{[y_i=j]} \log h_w(x_i)_j$ (cross-entropy loss) Now, we need to search for a $k \times d$ matrix W

$h_w(x_i)_j = \frac{e^{\langle w_j, x_i\rangle}}{\sum_{s=1}^{k} e^{\langle w_s, x_i\rangle}} \Rightarrow \frac{\partial h_w(x_i)_j}{\partial w_{st}} = \begin{cases} -h_w(x_i)_j \cdot h_w(x_i)_s \cdot x_{it} & \text{if } j \neq s \\ h_w(x_i)_j \cdot (1 - h_w(x_i)_s) \cdot x_{it} & \text{if } j = s \end{cases}$, $s \in [k]$ $t \in [d]$

$\frac{\partial L_S(h_w)}{\partial w_{st}} = \frac{1}{m}\sum_{i=1}^{m} (h_w(x_i)_s - \mathbb{1}_{[y_i=s]}) x_{it} = 0$

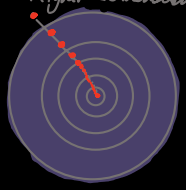
A set C in a vector space is convex if $\forall \vec{u},\vec{v} \in C$, the line segment between $\vec{u},\vec{v}$ is contained in C $\therefore \forall \alpha \in [0,1] : \alpha\vec{u} + (1-\alpha)\vec{v} \in C$

Let C convex. A function $f: C \to \mathbb{R}$ if $\forall u, v \in C$, $\alpha \in [0,1]$, $f(\alpha\vec{u} + (1-\alpha)\vec{v}) \leq \alpha f(\vec{u}) + (1-\alpha) f(\vec{v})$



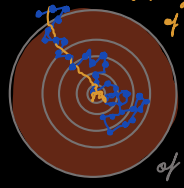
Gradient Descend: $w_{st} \leftarrow w_{st} - \alpha \frac{\partial L_S(h_w)}{\partial w_{st}}$, α: step size (Hyperparameter)

Higher dimensions



$\Longrightarrow$ Stochastic gradient descend: Don't compute the gradient of $L_S(h_w)$ exactly, instead use one or a batch of examples to estimate the gradient of $L_D(h_w)$ (i) Noisy but correct in expectation.

$\Longrightarrow$ Faster computation, Less overfitting



(SGD) for Logistic Reg:

Inputs: training examples S, step size α, batch size $b \leq |S|$

Initialize $w \in \mathbb{R}^{k \times d}$ randomly

converged $\leftarrow$ FALSE

while !converged:

"1 EPOCH"

- Shuffle S
- for $i = 0, \ldots, \lceil |S|/b \rceil - 1$:
  - $S' \leftarrow S[i \cdot b : (i+b) \cdot b]$
  - $w \leftarrow w - \alpha \nabla L_{S'}(h_w)$
- converged $\leftarrow$ check_convergence(S, w) △

return

(i) Convergence: $\|w^{(t+1)} - w^{(t)}\|_2 < \varepsilon$ (might waste time)

(ii) $|L_S(h_{w^{(t+1)}}) - L_S(h_{w^{(t)}})| < \varepsilon$. Approximately correct: $L_D(h_S) \leq \varepsilon$ $\varepsilon > 0$.

Since $S \sim D^m$: $\exists$ probability that S is not representative of D.

(Probability Approximately Correct PAC): $D^m(\{S|_x : L_D(h_S) > \varepsilon\})$

$S|_x = (x_1, x_2, \ldots, x_m)$ attribute instances of the training set.

(i) iid Data: $D^2(z_1,z_2) = D(z_1)D(z_2)$ $\forall z \in X \times Y$

(ii) Finite Hypothesis class: $|\mathcal{H}| < \infty$.

(iii) Realizability: $\exists h^* \in \mathcal{H} : L_D(h^*) = 0$

Training fails if: $h_{training} \in \mathcal{H}_B = \{h \in \mathcal{H} : L_D(h) > \varepsilon\}$

$L_D(h) > \varepsilon \iff D(h(x) \neq y > \varepsilon)$, $L_D(h) = \mathbb{E}_{x,y \sim D}[l(h(x),y)]$

Realizability $\Longrightarrow L_S(h_S) = 0$

Building a bound:

Let $\mathcal{M} = \bigcup_{h \in \mathcal{H}_B} \{S|_x : L_S(h) = 0\}$ (Set of training sets that are correctly classified by at least one bad hypothesis)

$\therefore D^m(\{S|_x : L_D(h_S) > \varepsilon\}) \leq D^m(\mathcal{M}) = D^m(\bigcup_{h \in \mathcal{H}_B} \{S|_x : L_S(h) = 0\})$, $\forall A, B \subset X$, $D(A \cup B) \leq D(A) + D(B)$

$\therefore D^m(\{S|_x : L_D(h_S) > \varepsilon\}) \leq D^m(\mathcal{M}) \leq \sum_{h \in \mathcal{H}_B} D^m(\{S|_x : L_S(h) = 0\})$

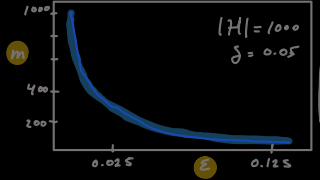
$D^m(\{S|_x : \forall i \in [m], h(x_i) = y_i\}) = \prod_{i=1}^{m} D(\{x_i : h(x_i) = y_i\})$ (iid), Note $D(\{x_i : h(x_i) = y_i\}) = 1 - \varepsilon$ $\therefore (\leq) \prod_{i=1}^{m} (1-\varepsilon)$

$\mathcal{H}_B = \{h \in \mathcal{H} : L_D(h) > \varepsilon\}$, $L_D(h) = \mathbb{E}_{x,y \sim D} \mathbb{1}[h(x) \neq y] \Longrightarrow D(\{x_i : h(x_i) = y_i\}) \leq 1 - \varepsilon$

$\leq \prod_{i=1}^{m} (1-\varepsilon) = (1-\varepsilon)^m$ i.e. $D^m(\{S|_x : L_D(h_S) > \varepsilon\}) \leq \sum_{h \in \mathcal{H}_B} D^m(\{S|_x : L_S(h) = 0\}) \leq \sum_{h \in \mathcal{H}_B} e^{-\varepsilon m} = |\mathcal{H}_B| e^{-\varepsilon m} \leq |\mathcal{H}| e^{-\varepsilon m}$

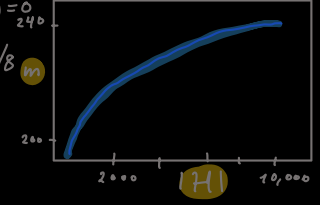
$\delta \geq |\mathcal{H}| e^{-\varepsilon m}$, $\log(\delta) \geq \log(|\mathcal{H}|) - \varepsilon m$ $\therefore 0 \geq \log\frac{|\mathcal{H}|}{\delta} - \varepsilon m \Longrightarrow m \geq \log\left(\frac{|\mathcal{H}|/\delta}{\varepsilon}\right)$

$m_{\mathcal{H}}(\varepsilon,\delta) \leq \left\lceil \frac{\log(|\mathcal{H}|/\delta)}{\varepsilon} \right\rceil$ Sample Complexity. $D^m(\{S|_x : L_D(h_S) > \varepsilon\}) \leq |\mathcal{H}| e^{-\varepsilon m}$



$\left\lceil \log\left(\frac{|\mathcal{H}|}{\delta}\right) / \varepsilon \right\rceil$

Theorem: (No-Free Lunch) Let A be any learning algorithm for the task of binary classification with respect to the 0-1 loss over a domain X. Let m be any number $m < \frac{|X|}{2}$ $\therefore \exists$ distribution D over $X \times \{0,1\}$ s.t. (i) $\exists f : X \to \{0,1\}$ s.t. $L_D(f) = 0$ (ii) With probability of at least $1/7$ over the choice $S \sim D^m$ we have $L_D(A(S)) \geq 1/8$



$\therefore$ A lower bound to PAC learning in binary-classification: $\frac{|X|}{2} \leq m_{\mathcal{H}}(\frac{1}{8}, \frac{1}{7})$

Corollary: Let X be an infinite domain set and let $\mathcal{H} = \{$all functions from $X \to \{0,1\}\}$. Then, $\mathcal{H}$ is not PAC learnable.

Error Decomposition: $L_D(h_S) = \varepsilon_{app} + \varepsilon_{est}$; ε_{app}: Approximation error (bias, quality of prev. knowledge).

$\varepsilon_{app} = \min_{h \in H} L_D(H)$, $\varepsilon_{est} = L_D(h_S) - \varepsilon_{app}$

Bias-Complexity Tradeoff: (i) Choosing H very rich class $\Rightarrow \varepsilon_{app} \downarrow$, $\varepsilon_{est} \uparrow$

(ii) Choosing H limited: $\Rightarrow \varepsilon_{est} \downarrow$, $\varepsilon_{app} \uparrow$

Model Selection with Validation: | TRAINING | VALIDATION | TESTING |

K-fold Cross Validation: (i) Split data into k subsets of equal size (ii) $\forall F_i$, $train\left(\bigcup_{i=1}^{k} F_i\right) \setminus F_a$, estimate error on F_a.

(3) Average error across all folds.

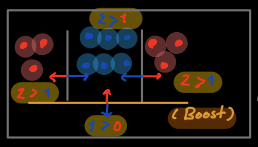
Regularization: Regularizer: $R: \mathcal{H} \to \mathbb{R}$, $h_S \in \operatorname{argmin}_{h \in \mathcal{H}} L_S(h) + R(h)$ (Regularized Loss Minimization)

L2 Regularization: $R(w) = \lambda \|w\|_2^2$, $\|w\|_2^2 = \sum_{i=1}^{d} w_i^2$

Ridge regression: linear / Polynomial + L_2 R. $\operatorname{argmin}_{w \in \mathbb{R}^d} \left(\lambda \|\vec{w}\|_2^2 + \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} (\langle \vec{w}, \vec{x_i} \rangle - y_i)^2 \right)$

What if we set $\lambda = 0$, and we still have high ε_{app}? **Ensemble:** Set of hypotheses, **Non-Realizability:** No hypothesis can correctly classify all data.

Separation with boosted half-spaces, **ML-Algorithm = representation + loss function + optimizer**

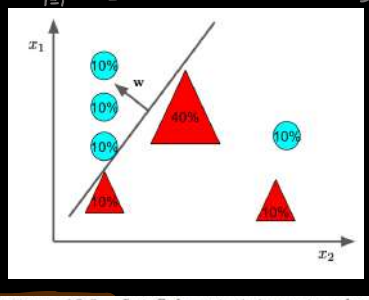


Let $\mathcal{H}$ be the base B (not-boosted hypothesis). Let $L(\mathcal{H}, T)$ be the class of ensemble hypotheses built using T elements of $\mathcal{H}$ ($L(B, T)$), $L(\mathcal{H}, T) = \{ \vec{x} \mapsto sign\left(\sum_{t=1}^{T} w_t h_t(\vec{x})\right) : \vec{w} \in \mathbb{R}^T, \forall t\ h_t \in \mathcal{H} \}$

(i) Our base hypothesis don't have to be PAC learnable, boosting works as long as they are γ-weakly-learnable: Reaching error $\frac{1}{2} - \gamma$ for fixed $0 < \gamma < 1/2$ instead of arbitrary $\varepsilon > 0$.

Representation: Distribution over Examples: We weight the m examples in S. Let $D^{(t)}$ be a distribution over the m examples in S for the t-th base learner. **Loss-Function:** $L_S(h_S) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{[h_S(\vec{x_i}) \neq y_i]}$ (0-1 loss)

$\therefore$ $\varepsilon_t \triangleq L_{D^{(t)}}(h_t) \triangleq \sum_{i=1}^{m} D_i^{(t)} \mathbb{1}_{[h_t(x_i) \neq y_i]}$, $D^{(t)} \in \mathbb{R}^m$



Optimizer: Ensemble

AdaBoost

input:
training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
weak learner WL
number of rounds T
initialize $\mathbf{D}^{(1)} = (\frac{1}{m}, \ldots, \frac{1}{m})$.
for $t = 1, \ldots, T$:
invoke weak learner $h_t = \mathrm{WL}(\mathbf{D}^{(t)}, S)$
compute $\epsilon_t = \sum_{i=1}^{m} D_i^{(t)} \mathbb{1}_{[y_i \neq h_t(\mathbf{x}_i)]}$
let $w_t = \frac{1}{2} \log\left(\frac{1}{\epsilon_t} - 1\right)$
update $D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(\mathbf{x}_i))}{\sum_{j=1}^{m} D_j^{(t)} \exp(-w_t y_j h_t(\mathbf{x}_j))}$ for all $i = 1, \ldots, m$
output the hypothesis $h_s(\mathbf{x}) = \mathrm{sign}\left(\sum_{t=1}^{T} w_t h_t(\mathbf{x})\right)$.

At first, example weights are all same

At each round, select a base hypothesis using example weights

Weight influence in the ensemble via its error

Update example weights via the ensemble's mistakes

THEOREM 10.2 *Let S be a training set and assume that at each iteration of AdaBoost, the weak learner returns a hypothesis for which $\epsilon_t \leq 1/2 - \gamma$. Then, the training error of the output hypothesis of AdaBoost is at most*

$$L_S(h_s) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{[h_s(\mathbf{x}_i) \neq y_i]} \leq \exp(-2\gamma^2 T) .$$

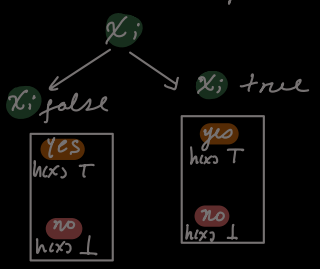
Decision Trees: **Boosting** is an algorithmic framework for extending a "base" hypothesis class to a more complex one $\mathcal{X} = \{T, \bot\}^{10}$, $\mathcal{X} = \{0,1\}^d$

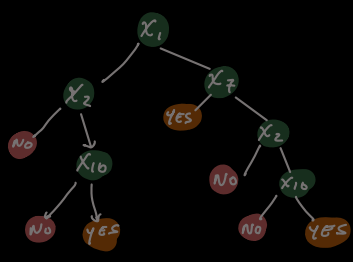
Decision Stump: Rule $(x_1 \wedge \neg x_7) \vee (x_2 \wedge x_{10})$

$\mathcal{Y} = \{0, 1\}$

$\mathcal{H} = \{h : h$ a decision Tree with layers $\leq T\}$

Implications: At most 2^T leaves, Complexity grows T.





(i) A decision Tree splits $(x, y) \in S$ into leaves. Let $l_S \subset S$ be the subset that reaches leaf l.

(ii) If we replace l with a split of x_i we get two new leaves $l_0 = \{(x, y) \in l_S \text{ s.t. } x_i = 0\}$

$l_1 = \{(x, y) \in l_S \text{ s.t. } x_i = 1\}$

Score of splitting a node l on x_i : $Gain(l,i,S) = C(\mathbb{P}_{x,y\sim l_S}[y=1]) - \mathbb{P}_{x,y\sim l_S}[x_i=1]\cdot C(\mathbb{P}_{x,y\sim l_S}[y=1|x_i=1])$
$- \mathbb{P}_{x,y\sim l_S}[x_i=0]\cdot C(\mathbb{P}_{x,y\sim l_S}[y=0|x_i=0])$, $C(a)=\min(a,1-a) \Rightarrow$ gain is improvement in training error.

Alternative node scores: Training error $C(a)=\min(a,1-a)$, Entropy: $C(a)=-a\log a-(1-a)\log(1-a)$

Gini Impurity: $C(a)=2a(1-a)$

Review: PAC Learning

- ***Probably approximately correct (PAC) learnability*** is a property of a hypothesis class $\mathcal{H}$. If it holds, there's some function that gives a number of i.i.d. training examples m that are sufficient to guarantee that $L_{\mathcal{D}}(h_S) \leq \epsilon$ with probability at least $1-\delta$ (for arbitrary ϵ and δ, and some algorithm)
- We've shown that any finite, realizable $\mathcal{H}$ is PAC learnable via ERM, with sample complexity

$$m_{\mathcal{H}}(\epsilon,\delta) \leq \left\lceil \frac{\log(|\mathcal{H}|/\delta)}{\epsilon} \right\rceil$$

Agnostic PAC Learning

- Property of a hypothesis class with respect to a data representation $\mathcal{X}\times\mathcal{Y}$ and loss ℓ, analogous to PAC, except relative to best hypothesis in the class
- There exists $m_{\mathcal{H}}:(0,1)^2\to\mathbb{N}$ and a learning algorithm such that, for any $\epsilon,\delta\in(0,1)$, if we have m i.i.d. examples where

$$m \geq m_{\mathcal{H}}(\epsilon,\delta)$$

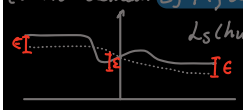
then with probability at least $1-\delta$, the learning algorithm returns h such that

$$L_{\mathcal{D}}(h) \leq \min_{h'\in\mathcal{H}} L_{\mathcal{D}}(h') + \epsilon$$

DEFINITION 4.3 (Uniform Convergence) We say that a hypothesis class $\mathcal{H}$ has the *uniform convergence property* (w.r.t. a domain Z and a loss function ℓ) if there exists a function $m_{\mathcal{H}}^{\mathrm{UC}}:(0,1)^2\to\mathbb{N}$ such that for every $\epsilon,\delta\in(0,1)$ and for every probability distribution $\mathcal{D}$ over Z, if S is a sample of $m\geq m_{\mathcal{H}}^{\mathrm{UC}}(\epsilon,\delta)$ examples drawn i.i.d. according to $\mathcal{D}$, then, with probability of at least $1-\delta$, S is ϵ-representative.

Bounded loss : $\exists a,b\in\mathbb{R}\ \forall h\in\mathcal{H}, \vec{x}\in\mathcal{X}, y\in\mathcal{Y}$
$\ell(h(x,y))\in[a,b]$

Uniform Convergence : Def: A training set S is ϵ-representative (w.r.t. domain $\mathcal{Z},\mathcal{H},\ell,\mathcal{D}$) if $\forall h\in\mathcal{H}, |L_S(h)-L_{\mathcal{D}}(h)|\leq\epsilon$



Lemma: Assume S: $\frac{\epsilon}{2}$-representative (w.r.t. $\mathcal{Z},\mathcal{H},\ell,\mathcal{D}$) Then, any output of $ERM_{\mathcal{H}}(S)$, namely $h_S\in\operatorname{argmin}_{h\in\mathcal{H}} L_S(h)$ satisfies
$L_{\mathcal{D}}(h_S)\leq\min_{h\in\mathcal{H}} L_{\mathcal{D}}(h)+\epsilon$ (ERM) (ε/2-rep)

Proof: $L_{\mathcal{D}}(h_S)\leq L_S(h_S)+\frac{\epsilon}{2}\leq L_S(h)+\frac{\epsilon}{2}\leq L_{\mathcal{D}}(h)+\frac{\epsilon}{2}+\frac{\epsilon}{2}$
$= L_{\mathcal{D}}(h)+\epsilon$

- We want to upper bound $\mathcal{D}^m(\{S:\exists h\in\mathcal{H}, |L_S(h)-L_{\mathcal{D}}(h)|>\epsilon\})$
- Observe that
$\{S:\exists h\in\mathcal{H}, |L_S(h)-L_{\mathcal{D}}(h)|>\epsilon\} = \cup_{h\in\mathcal{H}}\{S:|L_S(h)-L_{\mathcal{D}}(h)|>\epsilon\}$
- Then by the union bound
$\mathcal{D}^m(\{S:\exists h\in\mathcal{H}, |L_S(h)-L_{\mathcal{D}}(h)|>\epsilon\}) \leq \sum_{h\in\mathcal{H}}\mathcal{D}^m(\{S:|L_S(h)-L_{\mathcal{D}}(h)|>\epsilon\})$

Note $D^m(\{S:|L_S(h)-L_D(h)|>\varepsilon\})\longrightarrow 0$ as $m\longrightarrow\infty$.

Hoeffding Inequality: $\theta\in[a,b]$, $\theta\sim\mathbb{P}$, $E_{\mathbb{P}}[\theta]=\mu$ $\therefore\forall\varepsilon>0$ $\mathbb{P}\left[\left|\frac{1}{m}\sum_{i=1}^m\theta_i-\mu\right|>\varepsilon\right]\leq 2\exp\left(\frac{-2m\varepsilon^2}{(b-a)^2}\right)$
$\Rightarrow D^m(\{S:|L_S(h)-L_D(h)|>\epsilon\})\leq 2e^{-2m\epsilon^2}\Rightarrow D^m(\{S:\exists h\in\mathcal{H},|L_S(h)-L_D(h)|>\epsilon\})\leq\sum_{h\in\mathcal{H}}2\exp(-2m\varepsilon^2)=2|\mathcal{H}|e^{-2m\epsilon^2}$
$\therefore \ell:\mathcal{H}\times\mathcal{X}\times\mathcal{Y}\longrightarrow(0,1]$

Conclusions

- Any finite hypothesis class $\mathcal{H}$ has uniform convergence with respect to a loss $\ell:\mathcal{H}\times\mathcal{X}\times\mathcal{Y}\to[0,1]$ with sample complexity

$$m_{\mathcal{H}}^{\mathrm{UC}}(\epsilon,\delta)\leq\left\lceil\frac{\log(2|\mathcal{H}|/\delta)}{2\epsilon^2}\right\rceil$$

- Further, $\mathcal{H}$ is agnostically PAC learnable via ERM with sample complexity

$$m_{\mathcal{H}}(\epsilon,\delta)\leq m_{\mathcal{H}}^{\mathrm{UC}}(\epsilon/2,\delta)\leq\left\lceil\frac{2\log(2|\mathcal{H}|/\delta)}{\epsilon^2}\right\rceil$$

Comparison with PAC Learning

- Compare the sample complexity of PAC learning:

$$m_{\mathcal{H}}(\epsilon,\delta)\leq\left\lceil\frac{\log(|\mathcal{H}|/\delta)}{\epsilon}\right\rceil$$

with agnostic PAC learning:

$$m_{\mathcal{H}}(\epsilon,\delta)\leq m_{\mathcal{H}}^{\mathrm{UC}}(\epsilon/2,\delta)\leq\left\lceil\frac{2\log(2|\mathcal{H}|/\delta)}{\epsilon^2}\right\rceil$$

- Dropping realizability increases sample complexity by factor of more than $\frac{2}{\epsilon}$

Summary of Reasoning Steps

1. Assume we have a finite hypothesis class H and loss bounded in [0,1]
2. Then, H has uniform convergence
3. Then, with probability $1-\delta$, if we have a training sample S with size m, where

$$m_{\mathcal{H}}^{\mathrm{UC}}(\epsilon/2,\delta)\leq\left\lceil\frac{2\log(2|\mathcal{H}|/\delta)}{\epsilon^2}\right\rceil\leq m$$

then S is $\frac{\epsilon}{2}$- representative

4. If S is $\frac{\epsilon}{2}$- representative, then $L_{\mathcal{D}}(h_S)\leq\min_{h'\in\mathcal{H}}L_{\mathcal{D}}(h')+\epsilon$

The Most Important Things

- ***Agnostic probably approximately correct (PAC) learning*** is a property of a hypothesis class $\mathcal{H}$. If it holds, there's a function $m_{\mathcal{H}}:(0,1)^2\to\mathbb{N}$ and an algorithm such that if we have m i.i.d. examples where $m\geq m_{\mathcal{H}}(\epsilon,\delta)$, then with probability at least $1-\delta$ the algorithm returns h such that

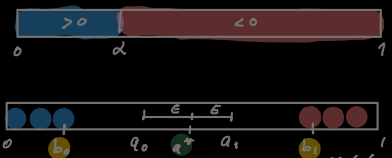
$$L_{\mathcal{D}}(h)\leq\min_{h'\in\mathcal{H}}L_{\mathcal{D}}(h')+\epsilon$$

- We've shown ***any finite hypothesis class is agnostic PAC learnable*** via ERM with respect to a loss function with range [0,1], with sample complexity

$$m_{\mathcal{H}}(\epsilon,\delta)\leq\left\lceil\frac{2\log(2|\mathcal{H}|/\delta)}{\epsilon^2}\right\rceil$$

Textbook: chapter 4

1-D Threshold



where a^* is the true threshold:

b_0: $\sup_i \{x_i : x \in S, x_i > 0\}$

b_1: $\inf_i \{x_i : x \in S, x_i < 0\}$

$D(x \in [a_0, a^*]) = \varepsilon$, $D^m(\{S : h_D(h_S) > \varepsilon\}) \le D^m(\{S : b_0 < a_0 \vee b_1 > a_1\})$

$D^m(\{S : b_0 < a_0 \vee b_1 > a_1\}) \le D^m(\{S : b_0 < a_0\}) + D^m(\{S : b_1 > a_1\}) = D^m(\{S : \#x_i \in (a_0, a^*)\}) + D^m(\{S : \#x_i \in (a^*, a_1)\})$

$= (1-\varepsilon)^m + (1-\varepsilon)^m \le 2e^{-\varepsilon m}$ $\therefore \forall \delta > (0,1)$, $m \ge \left\lceil \frac{\log(2/\delta)}{\varepsilon} \right\rceil \Rightarrow$ with probability $1-\delta$, $L_D(h_S) \le \varepsilon$

Shattering: A hypothesis class $\mathcal{H}$ shatters a finite set $C \subset \mathcal{X}$ if $\forall$ possible assignment of outputs to the points in C, $\exists h \in \mathcal{H}$ that induces it.

VC-Dimension: $VC(\mathcal{H})$ Maximal size set C s.t. $\mathcal{H}$ shatters C. The VC dimension of the class of homogeneous halfspaces in $\mathbb{R}^d$ is d

The Fundamental Theorem of Statistical Learning

Let H be a hypothesis class of functions from a domain X to {0, 1} and let the loss function be the 0−1 loss. Then, the following are equivalent:

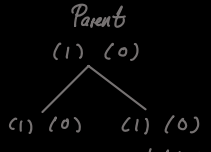
1. H has the uniform convergence property
2. H is agnostic PAC learnable
3. The ERM algorithm is a successful agnostic PAC learner for H
4. H is PAC learnable
5. The ERM algorithm is a successful PAC learner for H
6. H has a finite VC-dimension

- We can prove that a hypothesis class has VC dimension of d by showing
  - (i) $\exists C \subset X$ s.t. $|C| = d$ and C shattered by $\mathcal{H}$
  - (ii) $\exists C' \subset X$ s.t. $|C'| = d+1$ C' not shattered by $\mathcal{H}$

Weak learners: A class C is weakly PAC learnable using a hypothesis class $\mathcal{H}$ if $\exists$ algorithm A, $\exists \gamma > 0$ s.t. $\forall c \in C$ $\forall D$ distribution over the input space, $\forall \delta \in (0, 1/2)$, given access to a polynomial (in $1/\delta$) number of examples drawn i.i.d from D and labeled by f, A outputs a function $h \in \mathcal{H}$ s.t. with $P \ge 1-\delta$, $L_{(D,f)}(h) \le 1/2 - \gamma$

Strong learners: A class C is strongly PAC learnable using $\mathcal{H}$ if $\exists A$ s.t. $\forall c \in C$, $\forall D$ over S $\forall \varepsilon \in (0, 1/2)$, $\delta \in (0, 1/2)$, given access to a polynomial (in $1/\varepsilon$, $1/\delta$) number of examples drawn i.i.d. from D and labeled as f, A outputs a function $h \in \mathcal{H}$ s.t. with $P \ge 1-\delta$, $L_{(D,f)}(h) \le \varepsilon$

Naive Bayes: Collect all words in training data into vocab. set. V = "Milk" "Chocolate" "Toffee" "almond"... where $|V| = d$ (assume the set is ordered s.t. each word has index $i \in [d]$. Then $X = \{0,1\}^d$ i.e. "Milk Chocolate" $= (1, 0, 1, 0, 0, \ldots)$. Representation: Hypothesis class: Instead of learning a hypothesis $h: X \to Y$ we learn a distribution $P_\theta(x, y)$ that defines a hypothesis $P_\theta(y|x)$. $P_\theta(X=1) = \theta^x (1-\theta)^{1-x}$, $\prod_{i=1}^m P_\theta(x_i = 1)$



$\begin{cases} P(\text{parent} = 1) = p_{\text{parent}} \\ P(A = 1) = p_A \\ P(B = 1) = p_B \end{cases}$, $\begin{cases} p_{1,1} = p_{\text{parent}} \cdot p_A \\ p_{1,0} = p_{\text{parent}} \cdot (1 - p_A) \\ p_{0,1} = (1 - p_{\text{parent}}) \cdot p_B \end{cases}$

$P_\theta(x, y) = P_\theta(y) \prod_{i=1}^d P_\theta(x_i | y)$ **Generative Story**: (a) Randomly generate a class (b) Based on the class, select a set of d conditionally independent "child" distributions (c) Generate each attribute, using the corresponding "child" distribution for each. **Maximum Likelihood Estimator**: $\ell(\theta, (x, y)) = -\log[P_\theta(x, y)]$ (log loss). Maximum likelihood: $\hat{\theta} \in \arg\min_\theta \sum_{i=1}^m -\log[P_\theta(x_i, y_i)]$

$P_\theta(y)$: Prior Probability (i) What is $P_\theta(y)$ without knowing parameters

$P(y|x) = \frac{P(x|y) P(y)}{P(x)}$ (Bayes theorem)

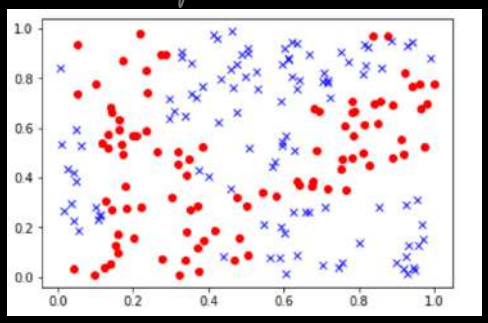
$= \frac{P(y) \prod_{i=1}^d P(x_i|y)}{P(x)}$ (Naive Bayes Assumption)

$\propto P(y) \prod_{i=1}^d P(x_i|y)$

Note: $\prod_{i=1}^d P_\theta(x_i|y) = \exp\left(\log\left(\prod_{i=1}^d P_\theta(x_i|y)\right)\right)$

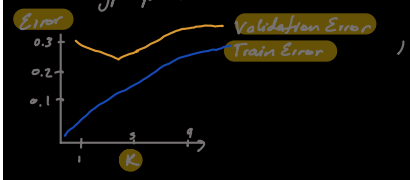
$= \exp\left(\sum_{i=1}^d \log P_\theta(x_i|y)\right)$

K-Nearest Neighbors:



Distance Functions If $\mathcal{X} = \mathbb{R}^d$, $\delta(x_1, x_2) = \sqrt{\sum_{j=1}^{d} [(x_1)_j - (x_2)_j]^2}$

- K: hyperparameter $\therefore$ use model selector curve or cross validation.



, $K=1 \Longrightarrow VC = \infty \Longrightarrow$ ⊬ PAC learnable

Margin: Minimum distance between any training set and decision boundary

the distance between point (a) to decision boundary $\langle w, x \rangle = 0$ is $\frac{|\langle w, a \rangle|}{\|w\|}$ $\therefore$ Margin: $\min_{i \in [m]} \frac{|\langle w, x_i \rangle|}{\|w\|}$ set $\|w\|=1$

Support Vector Machines: (SVM): algorithm for learning a halfspace with maximum margin. (Binary Classification). Assume: (i) Realizable (i.e. lin. separable). "Hard SVM": $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1, 1\}$, $h_w(x) = \text{sign}(\langle w, x \rangle)$, $\vec{w}$ chosen to minimize empirical risk, We assume $L_S(h) = 0$ (hard)

goal $\therefore \operatorname{argmax}_{w: \|w\|=1} \min_{i \in [m]} |\langle w, x_i \rangle|$ s.t. $\forall i \; y_i \langle w, x_i \rangle > 0$

(i) $\hat{w} \triangleq \operatorname{argmax}_{w: \|w\|=1} \min_{i \in [m]} y_i \langle w, x_i \rangle \Longrightarrow \forall i, y_i \langle \hat{w}, x_i \rangle > 0$, $\forall i, y_i \langle \hat{w}, x_i \rangle > 0 \Longrightarrow \min_{i \in [m]} y_i \langle \hat{w}, x_i \rangle = \min_{i \in [m]} |\langle \hat{w}, x_i \rangle|$ $\therefore$ We rewrite

$\operatorname{argmax}_{w: \|w\|=1} \min_{i \in [m]} y_i \langle w, x_i \rangle \Longleftrightarrow \operatorname{argmax}_{M, w: \|w\|=1} M$ s.t. $\forall i \in [m], \; y_i \langle w, x_i \rangle \geq M$, M (greatest lower bound)

$\Longleftrightarrow \operatorname{argmax}_{M, w: \|w\|=1} M$ s.t. $\forall i \in [m], \; y_i \langle \frac{w}{M}, x_i \rangle \geq 1 \Longleftrightarrow \operatorname{argmin}_{M, w: \|w\|=1} \frac{1}{M}$ s.t. $\forall i \in [m], \; y_i \langle \frac{w}{M}, x_i \rangle \geq 1$

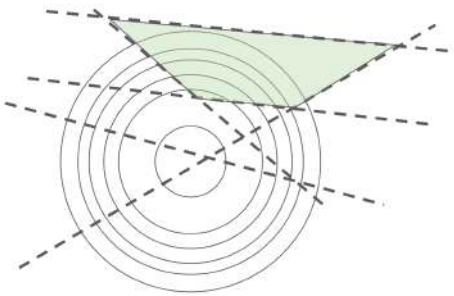
$\Longleftrightarrow \operatorname{argmin}_{w'} \|w'\|$ s.t. $\forall i \in [m], \; y_i \langle w', x_i \rangle \geq 1$ $w' = \frac{w}{M}$ $\therefore$ the halfspaces defined by w and w' are identical

$\Longleftrightarrow \operatorname{argmin}_{w} \|w\|^2$ s.t. $\forall i \in [m], \; y_i \langle w, x_i \rangle \geq 1$

- Our rewritten objective can be visualized as:

$$\arg\min_{\mathbf{w}} \; \|\mathbf{w}\|^2$$

$$\text{s.t. } \forall i \in [m], \quad y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1$$



Theoretical Guarantee

THEOREM 15.4 *Let $\mathcal{D}$ be a distribution over $\mathbb{R}^d \times \{\pm 1\}$ that satisfies the (γ, ρ)-separability with margin assumption using a homogenous halfspace. Then, with probability of at least $1 - \delta$ over the choice of a training set of size m, the 0-1 error of the output of Hard-SVM is at most*

$$\sqrt{\frac{4\,(\rho/\gamma)^2}{m}} + \sqrt{\frac{2\log(2/\delta)}{m}}.$$

(Contrast with guarantee provided by fundamental theorem of statistical learning which says error $\leq \mathcal{O}\left(\sqrt{d}\right)$)

Soft SVM: $\nexists w \in \mathbb{R}^d$ s.t. $\forall i \in [m], \; y_i \langle w, x_i \rangle \geq 1$, $\xi_i \triangleq$ How far over the margin boundary is

$\operatorname{argmin}_{w, \xi} \lambda \|w\|^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i$ s.t. $\forall i \in [m] \; y_i \langle w, x_i \rangle \geq 1 - \xi_i$ and $\xi_i \geq 0$

Solving Soft SVM with Gradient Descent

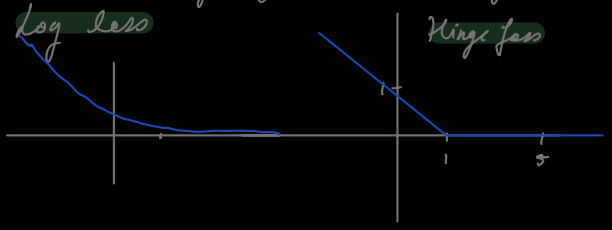
$$\arg\min_{\mathbf{w}} \; \lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^{m} \max\left(0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle\right)$$

- Optimization objective is convex!
- Differentiable everywhere except at $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle = 1$ for any $\mathbf{x}_i$
  - The gradient of that term can just be treated as zero (subgradient)

Last Time

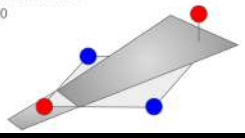
- A ***support vector machine*** (SVM) is an algorithm for learning a halfspace with maximum margin
- **Hard** SVM assumes that the data are realizable with a halfspace
- By assuming that the data is separated by a wide margin, we get sample complexity that doesn't depend on input dimension
- **Soft** SVM relaxes that assumption and penalizes how far data points are on the wrong side of the decision boundary

Soft SVM = Regularized ERM with hinge loss: $\operatorname{argmin}_{w} \lambda \|w\|^2 + \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y_i \langle w, x_i \rangle)$



Why Increase Input Dimensionality?

- Consider XOR:
  - $(0,0) \to 0$, $(0,1) \to 1$, $(1,0) \to 1$, $(1,1) \to 0$
  - No linear separator.
- What happens if we add $x_1 x_2$ as a separate feature?
  - $(0,0,0) \to 0$, $(0,1,0) \to 1$, $(1,0,0) \to 1$, $(1,1,1) \to 0$
- Now, it's linearly separable!
  - $w = (1,1,-2)$, $b = -½$



i.e. $\psi : \mathbb{R}^2 \to \mathbb{R}^3$, $\psi(x_1, x_2) = (x_1, x_2, x_1 x_2)$

ex: Polynomials up to degree K:

$\psi(x_1, x_2) = (1, \sqrt{3}x_1, \sqrt{3}x_2, \sqrt{6}x_1x_2, \sqrt{3}x_1^2, \sqrt{3}x_2^2, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_1^3, x_2^3)$

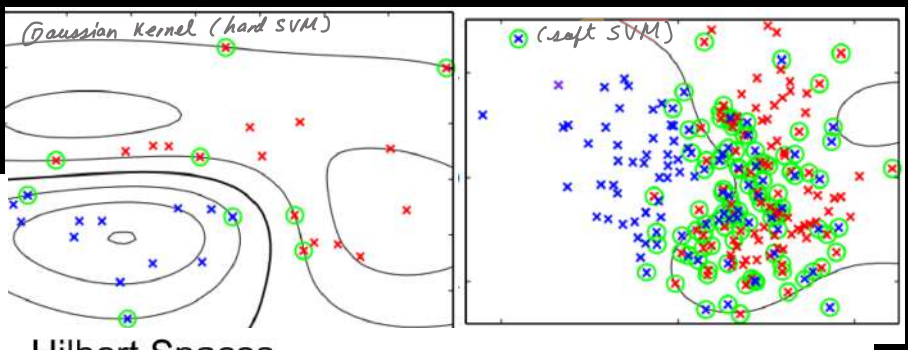
Kernel Trick: $\langle \psi(x_i), \psi(x_j) \rangle$, $K(x_i, x_j) = \langle \psi(x_i), \psi(x_j) \rangle$

Polynomial Kernel: $K(x_i, x_j) = (1 + \langle x_i, x_j \rangle)^k$

Gaussian Kernel: $K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2}}$ (feature space of infinite dimension)

$\forall n \geq 0 : \psi(x)_n = \frac{1}{\sqrt{n!}} e^{-x^2/2} x^n$

Redefine the optimal solution w^* as a weighted combination of examples: $w^* = \sum_{i=1}^{m} \alpha_i \psi(x_i)$



Representer Theorem

Statement: $\mathbf{w}^\star = \sum_{i=1}^{m} \alpha_i \psi(\mathbf{x}_i)$ for some values $\boldsymbol{\alpha} \in \mathbb{R}^m$ where $\psi : \mathbb{R}^d \to \mathcal{F}$ is a feature embedding and $\mathcal{F}$ is a Hilbert space

Rewriting the Weight Norm

- Now we know that we can search for the optimal $\boldsymbol{\alpha}^\star$ that defines

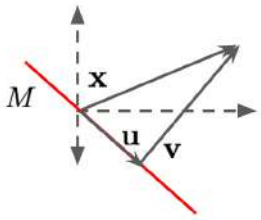
$$\mathbf{w}^\star = \sum_{i=1}^{m} \alpha_i^\star \psi(\mathbf{x}_i)$$

- We can rewrite the norm $\|\mathbf{w}\|^2$ in our soft SVM objective as:

$$\|\mathbf{w}\|^2 = \left\langle \sum_j \alpha_j \psi(\mathbf{x}_j), \sum_j \alpha_j \psi(\mathbf{x}_j) \right\rangle = \sum_{i,j=1}^{m} \alpha_i \alpha_j \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \rangle$$
$$= \sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Hilbert Spaces

- Type of vector space (including infinite dimensions)
  - Technical details: has inner product and is complete, i.e., all Cauchy sequences converge
- Useful fact: any point $\mathbf{x}$ in a Hilbert space can be rewritten as the sum $\mathbf{x} = \mathbf{u} + \mathbf{v}$ for any linear subset M of the Hilbert space where $\mathbf{u} \in M$ and $\langle \mathbf{v}, \mathbf{w} \rangle = 0 \ \ \forall \mathbf{w} \in M$



Rewriting the Hinge Loss

$$\frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y_i \langle \mathbf{w}, \psi(\mathbf{x}_i) \rangle)$$
$$= \frac{1}{m} \sum_{i=1}^{m} \max\left(0, 1 - y_i \left\langle \left(\sum_{j=1}^{m} \alpha_j \psi(\mathbf{x}_j) \right), \psi(\mathbf{x}_i) \right\rangle\right)$$
$$= \frac{1}{m} \sum_{i=1}^{m} \max\left(0, 1 - y_i \sum_{j=1}^{m} \alpha_j \langle \psi(\mathbf{x}_j), \psi(\mathbf{x}_i) \rangle\right)$$
$$= \frac{1}{m} \sum_{i=1}^{m} \max\left(0, 1 - y_i \sum_{j=1}^{m} \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)\right)$$

Define Gram Matrix $G \in \mathbb{R}^{m \times m}$, $G_{ij} = K(x_i, x_j)$

$\therefore \min_{\alpha \in \mathbb{R}^m} \lambda \alpha^T G \alpha + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i (G\alpha)_i\}$

Predictions:

Take the sign of $\langle w, \psi(x) \rangle = \sum_{j=1}^{m} \alpha_j \langle \psi(x_j), \psi(x) \rangle = \sum_{j=1}^{m} K(x_j, x)$

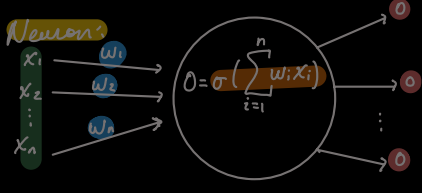
String Kernel

Represent a string as a vector by creating a dimension for every possible substring and putting in the count of how many times that sequence appears in the string. So, the vector for "banana" has:

$v_b = 1, v_a = 3, v_n = 2, v_{ba} = 1, v_{an} = 2, v_{na} = 2, v_{ban} = 1, v_{ana} = 2, v_{nan} = 1, v_{bana} = 1, v_{anan} = 1, v_{nana} = 1, v_{banan} = 1, v_{anana} = 1, v_{banana} = 1$, otherwise, $v_i = 0$

Infinite dimensional! But, can compute the kernel value efficiently: How many substrings are in common between two strings?

Neural Networks:



σ: Activation Function (Non-linear)

Step: $\sigma(a) = 1$ if $a > 0$, 0, otherwise

Sigmoid: $\sigma(a) = \frac{1}{(1 + \exp^{-a})}$

ReLU: $\sigma(a) = a$ if $a > 0$,

Goal: learn w_i

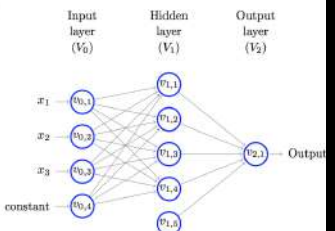
Representational Power:

(i) Using step functions and Network with one (giant) hidden layer, can capture any Boolean Function

Feedforward Neural Networks

Neurons made into an acyclic graph

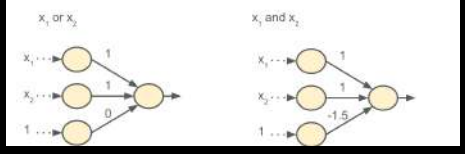
- Neurons with no incoming edges are input (first) layer
- Neurons with no outgoing edges are output (last) layer



NNs Can Represent Any Boolean Function

Assume for now: $\mathcal{X} = \{0,1\}^d$, $\mathcal{Y} = \{0,1\}$, and $\sigma(a) = 1$ if $a > 0$, 0 otherwise

Examples



(ii) Can convert poly-time computations into poly-size networks

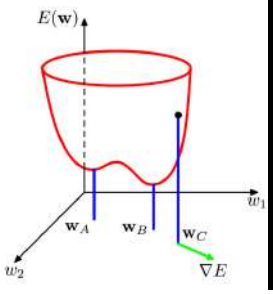
(iii) Universal Approximation for regression problems

Learning NN: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{X} = \mathbb{R}^{w \times h \times 3}$ ← Input

Output: $\mathcal{Y} = \{0,1\}$, Regression $\mathcal{Y} = \mathbb{R}$ ← Output

Hypothesis Class: Assume Architecture structure is fixed (including σ's) and learning is over all weights. Optimizer: SGD

Loss is Non Convex in General



Neural Networks are hypothesis classes that perform simple computations on input data, ending in a prediction and defined by learned parameters.

SGD for Neural Networks:

Stochastic Gradient Descent for Neural Networks

Inputs: training examples S, step size α, batch size $b < |S|$

Initialize collection of all weights $\mathbf{W}$ randomly

converged ← False

while !converged:

  Shuffle S

  for i = 0 … $\lceil |S| / b \rceil - 1$:

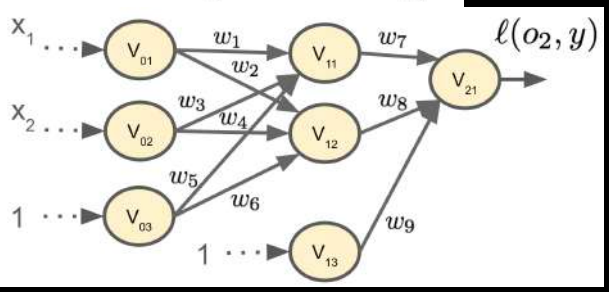
    $S' \leftarrow S[i \cdot b : (i+1) \cdot b]$

    $\mathbf{W} \leftarrow \mathbf{W} - \alpha \nabla L_{S'}(h_{\mathbf{W}})$

  converged ← check_convergence$(S, \mathbf{W})$

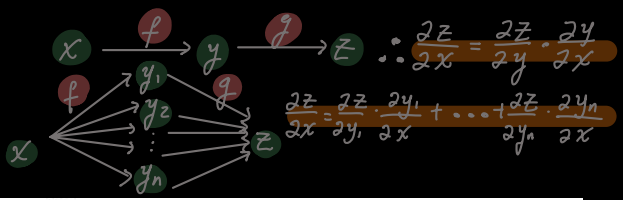
return

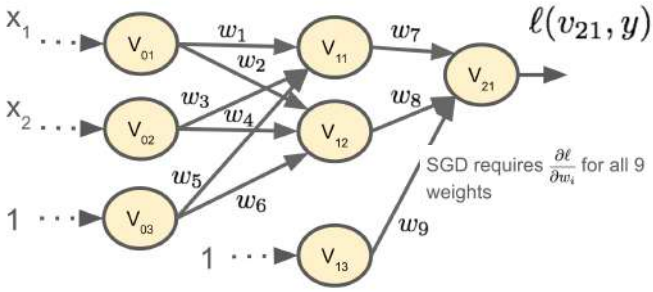
Recall that $\nabla L_{S'}(h_{\mathbf{W}})$ is the average of $\nabla \ell(h_{\mathbf{W}}(\mathbf{x}), y)$ over each of the $(\mathbf{x}, y)$ in S'



Chain Rule: $x, y, z \in \mathbb{R}$, $z = g(y)$, $y = f(x)$, $z = g(f(x))$

Multivariate Chain Rule: $x, z \in \mathbb{R}$, $y \in \mathbb{R}^n$, $z = g(y)$, $y = f(x)$



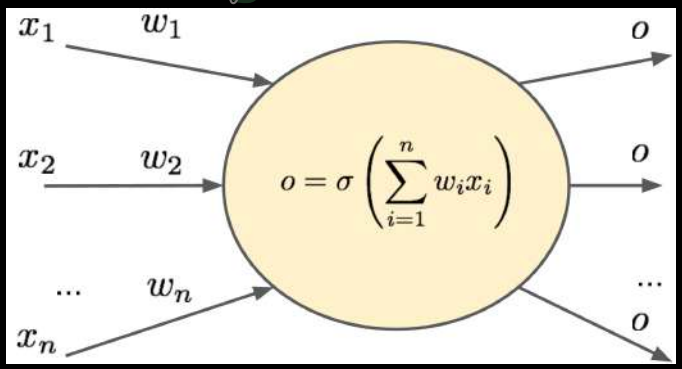


Derivatives of Activation Functions

Step: $\sigma(a) = 1$ if $a > 0$, 0 otherwise; $\frac{\partial \sigma}{\partial a} = 0$

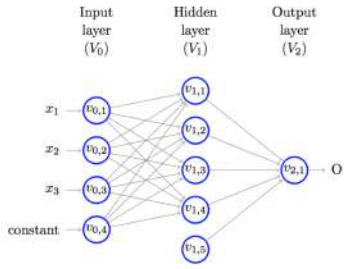
ReLU: $\sigma(a) = a$ if $a > 0$, 0 otherwise; $\frac{\partial \sigma}{\partial a} = 1$ if $a > 0$, 0 otherwise.

Sigmoid: $\sigma(a) = 1/(1 + \exp(-a))$; $\frac{\partial \sigma}{\partial a} = \sigma(a)(1 - \sigma(a))$

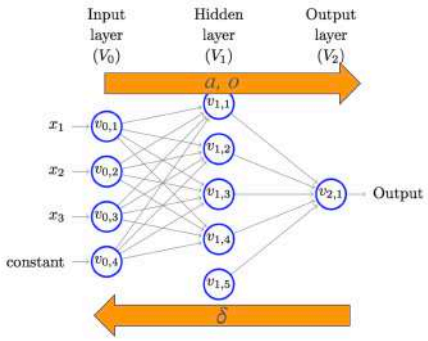


Notation

- Let $W_{j,j+1}$ be the matrix of weights connecting layers j and j+1 of dimension outputs x inputs
- b_j is the vector of weights on the bias term going into layer j
- Let $u \odot v$ be element-wise multiplication of vectors u and v



Backpropagation



As you send δ back, combine with a and o to compute all

$$\frac{\partial \ell}{\partial w}$$

Backpropagation

1. Compute all a, o (forward pass)
2. Set $\delta_n = \frac{\partial \ell}{\partial o_{n,1}} \sigma'(a_{n,1})$ where $a_{n,1}$ is input to last neuron
3. For layer j = n-1 down to 0:

$$\frac{\partial \ell}{\partial W_{j,j+1}} = \delta_{j+1} \cdot o_j^\top$$

$$\frac{\partial \ell}{\partial b_{j+1}} = \delta_{j+1}$$

$$\delta_j = (W_{j,j+1}^\top \cdot \delta_{j+1}) \odot \sigma'(a_j)$$

- The backwards messages have a precise interpretation:

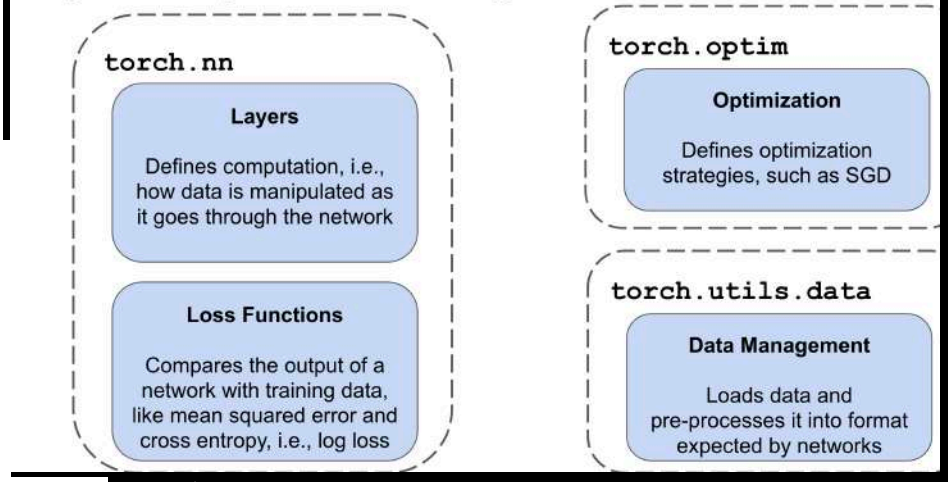
$$\delta_j = \frac{\partial \ell}{\partial a_j}$$

- Then

$$\frac{\partial \ell}{\partial W_{j-1,j}} = \frac{\partial \ell}{\partial a_j} \cdot \frac{\partial a_j}{\partial W_{j-1,j}} = \delta_j \cdot o_{j-1}^\top$$

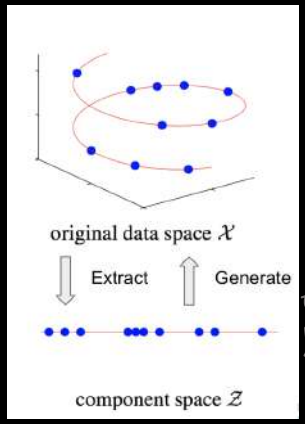
$$\begin{aligned}
\frac{\partial \ell}{\partial W_{0,1}} &= \delta_1 \cdot o_0^\top \\
&= \left((W_{1,2}^\top \cdot \delta_2) \odot \sigma'(W_{0,1} \cdot \mathbf{x} + b_0)\right) \cdot \mathbf{x}^\top \\
&= \left((W_{1,2}^\top \cdot \frac{\partial \ell}{\partial o_{2,1}} \cdot \sigma'(W_{1,2} \cdot o_1 + b_1)) \odot \sigma'(W_{0,1} \cdot \mathbf{x} + b_0)\right) \cdot \mathbf{x}^\top
\end{aligned}$$

Key Concepts and Packages




```
# Imports torch
import torch
from torch import nn

# Defines our first network
class FirstNetwork(nn.Module):
    def __init__(self, multiplier):
        super().__init__()
        self.multiplier = nn.Parameter(torch.tensor(float(multiplier)))

    def forward(self, x):
        return self.multiplier * x

# Initializes FirstNetwork
network = FirstNetwork(2)
print(network)

# Runs the network on an input and gets the result
network(4).item()
```

K-Means: Unsupervised learning: • Discovering structure in data. Supervised: (1) Given input/output examples, find mapping (2) Predictive: What will happen, What is missing. Unsupervised: Given data, finds a representation. Descriptive: What happened
Clustering: Groups of related examples



Dimensionality Reduction:
Unsupervised Algorithm: No class labels
"Hypothesis class" is now possible structures in data", Loss: Measures quality of "fit" of structure to data. Optimizer: Seeks the structure from the hypothesis class with best fit.

K-means clustering: Group data that are close into a cluster represented by their mean. Loss function defined on assignment of data to clusters (Group data to minimize loss). K is a hyperparameter

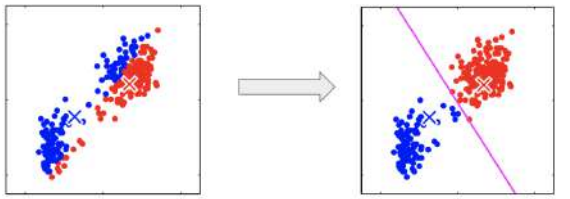
Partition example into $C_1, \dots, C_K$. Each Cluster has a centroid $\forall i: \mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$. Find clustering s.t. loss is minimized: $\sum_{i=1}^{K} \sum_{x \in C_i} \|\mu_i - x\|_2$. NP-hard to minimize. Powerful Greedy Technique can be used: Randomly choose initial centroids: $\mu_i = x \sim S$

Repeat until convergence: $\forall i: C_i = \{x \in S : i = \arg\min_j \|x - \mu_j\|_2^2\}$ (Use centroids to define clusters)

$\forall i: \mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$ (Use clusters to define centroids)

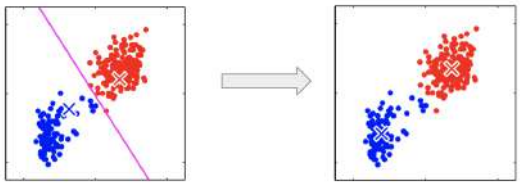
Use Centroids to Define Clusters

- $\forall i: C_i = \{x \in S : i = \arg\min_j \|x - \mu_j\|_2^2\}$
- Update: keep centroids fixed and (possibly) reassign clusters for data



Use Clusters to Define Centroids

- $\forall i: \mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$
- Update: keep clusters fixed and (possibly) recompute center for each C_i



K Means Optimizer Converges in Finite Time

- Infinite number of possible centers, but finite set of possible clusterings: K^S
- Stop iterating when criterion fails to improve
- Since each iteration has an associated clustering and each iteration improves the criterion, once we leave a clustering, we can never return to it
- Eventually must exhaust all clusterings
- But, could be many iterations or get stuck before reaching global minimum

Distribution over cluster Membership:

$\mathcal{Z} = \{1, \dots, K\}$ set of cluster identifiers. Let $z_i \in \mathcal{Z}$ a random variable denoting the cluster to which x_i belongs.

∴ We try to learn a distribution $p(X, Z)$ that will tell us $p(z_i | x_i)$

Multivariate Gaussians

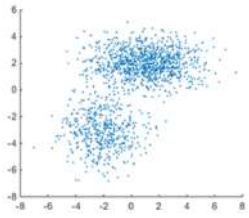
- Probability density function:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = (2\pi)^{-\frac{d}{2}} \det(\Sigma)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$

- Examples: $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ Σ: Covariance Matrix $\Sigma = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 1 \end{bmatrix}$

Mixture of K-Gaussians: (Generative Model for clustering)

Mixtures of K Gaussians

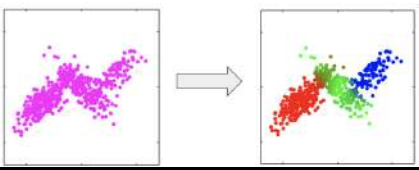


- Parameters:
  - θ_C defining multinomial prior distribution over cluster membership
  - K cluster centers $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$
  - K cluster covariance matrices $\Sigma_1, \dots, \Sigma_K$
- Generative story:
  - Pick a cluster $\boldsymbol{z}$ using $p_{\theta_C}(z)$
  - Generate attributes using multivariate Gaussian defined by $\boldsymbol{\mu}_z$ and Σ_z

Predicting Cluster Membership

- If we have all the model parameters, then we can predict $p(z_i|\mathbf{x}_i)$, the distribution over which cluster each point belongs to, using Bayes' Theorem:

$$p(z_i = j|\mathbf{x}_i) \propto p_{\theta_C}(z_i = j)\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_j, \Sigma_j)$$



Pattern Recognitio
Machine Learning
Bishop, 2006

Learning Mixtures of Gaussians with Fixed Clusters

- If we knew which cluster each example belonged to, i.e., we knew all the values of Z, then we could maximize the joint log likelihood of X and Z:

$$\hat{\theta}_C, \hat{\boldsymbol{\mu}}, \hat{\Sigma} = \arg\max_{\theta_C, \boldsymbol{\mu}, \Sigma} \sum_{i=1}^{m} \log p_{\theta_c}(z_i)\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_{z_i}, \Sigma_{z_i})$$

- Learning is just like Naive Bayes:
  - Set θ_C to the fractions of examples in each cluster
  - Set $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$ and $\Sigma_1, \dots, \Sigma_K$ to the mean and covariances of each cluster

Learning with Latent Variables

- Would that it were so simple…
- We don't know any of the values of Z! (They're ***latent***.)
- Instead, what we'd like to do is maximize the log ***marginal*** likelihood of X:

$$\hat{\theta}_C, \hat{\boldsymbol{\mu}}, \hat{\Sigma} = \arg\max_{\theta_C, \boldsymbol{\mu}, \Sigma} \sum_{i=1}^{m} \log p(\mathbf{x}_i)$$

$$= \arg\max_{\theta_C, \boldsymbol{\mu}, \Sigma} \sum_{i=1}^{m} \log \sum_{j=1}^{K} p_{\theta_c}(z_i = j) \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_j, \Sigma_j)$$

What's Hard about the Log Marginal Likelihood?

Think about solving the following optimization problem:

$$\hat{\theta}_C, \hat{\boldsymbol{\mu}}, \hat{\Sigma} = \arg\max_{\theta_C, \boldsymbol{\mu}, \Sigma} \sum_{i=1}^{m} \log \sum_{j=1}^{K} p_{\theta_c}(z_i = j) \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_j, \Sigma_j)$$

Which challenges make it harder than maximizing the joint log likelihood?

A: Does not decompose into separate learning problems over each cluster

B: Marginal likelihood is not differentiable

C: Cannot work directly with log likelihood of each example

D: A and B

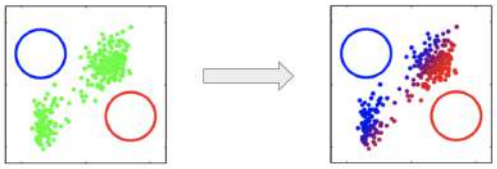
E: A and C

EM for Mixtures of K Gaussians

- Similar to K-Means!
- Initialize cluster prior uniformly: $p_{\theta_C}(z = j) = \frac{1}{K}$
- Randomly choose initial means with identity covariance: $\boldsymbol{\mu}_j = \mathbf{x} \sim S, \ \Sigma_j = I$
- Repeat until convergence:
  - E Step — Use Gaussians to define clusters
  - M Step — Use clusters to define Gaussians
- Eventually, no improvements are made

E Step

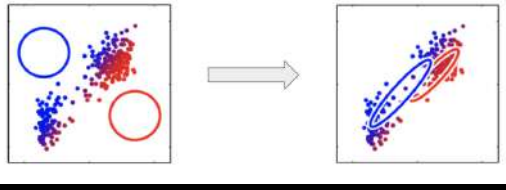
- $\gamma(z_{ij}) = \frac{p_{\theta_C}(z_i = j) \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_j, \Sigma_j)}{\sum_{k=1}^{K} p_{\theta_C}(z_i = k) \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \Sigma_k)}$
- Update: keep Gaussians and cluster prior fixed and update clusters $p(z_i | \mathbf{x}_i)$



Pattern Recognition and

M Step

- $N_j = \sum_{i=1}^{m} \gamma(z_{ij}), \quad \boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{i=1}^{m} \gamma(z_{ij}) \mathbf{x}_i, \quad \Sigma_j = \frac{1}{N_j} \sum_{i=1}^{m} \gamma(z_{ij}) (\mathbf{x}_i - \boldsymbol{\mu}_j)(\mathbf{x}_i - \boldsymbol{\mu}_j)^\top, \quad p_{\theta_C}(z = j) = \frac{N_j}{m}$
- Update: keep clusters $p(\boldsymbol{z}_i | \mathbf{x}_i)$, and update Gaussians and cluster prior

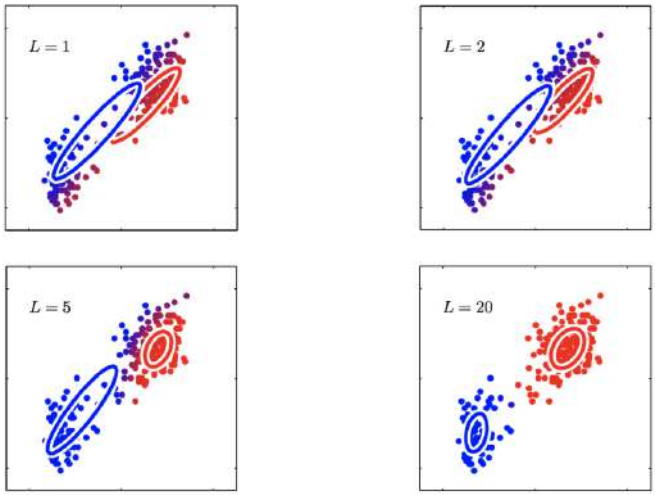


Pattern Recognition and

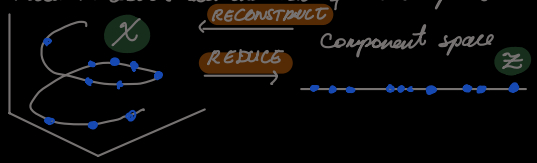
A Tractable Lower Bound

- Rather than work with the marginal distribution $p(X|\theta)$ (which requires marginalizing out Z) directly, we will work with a surrogate distribution $q(Z)$
- For a given set of parameters θ, we can measure the divergence from $q(Z)$ to $p(Z|X, \theta)$ with the Kullback-Leibler (KL) divergence:

$$\mathrm{KL}(q \| p) = -\sum_{Z} q(Z) \log \left\{ \frac{p(Z|X, \theta)}{q(Z)} \right\}$$



Dimensionality Reduction (PCA): transformation to new attributes in lower dimensional feature space



$\mathcal{X} = \mathbb{R}^d$, $\mathcal{Z} = \mathbb{R}^n$ $(n < d)$. We will learn $W: \mathcal{X} \to \mathcal{Z}$, $W \in \mathbb{R}^{n \times d}$, $z = Wx$

Define a loss function: $x \xrightarrow{W} z \xrightarrow{U} x'$. $d(x, h(x)) = \|x - x'\|_2^2$, $W, U = \underset{W,U}{\operatorname{argmin}} \sum_{i=1}^{m} \|x_i - UWx_i\|_2^2$

$W \in \mathbb{R}^{n \times d}$, $U \in \mathbb{R}^{d \times n}$

PCA:

Implications

- So far we've made a few key decisions:
  - Linear transformation for the reduction
  - Linear transformation for the reconstruction
  - Squared error loss function

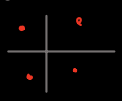
PCA Objective

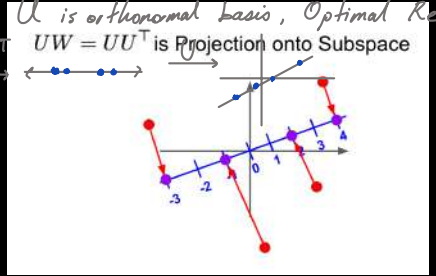
$$W, U = \arg\min_{W,U} \sum_{i=1}^{m} \|\mathbf{x}_i - UW\mathbf{x}_i\|_2^2$$

$$W \in \mathbb{R}^{n \times d} \qquad U \in \mathbb{R}^{d \times n}$$

- Just these decisions have big implications:
  - Columns of optimal reconstruction U are orthonormal basis for a subspace
  - Optimal reduction $W = U^T$
  - $UW = UU^\top$ is an orthogonal projection onto that subspace

$UW = UU^\top$, U is orthonormal basis, Optimal Reduction $W = U^\top$





Proof Sketch (Lemma 23.1)

- Pick any U, W and consider $\mathbf{x} \mapsto UW\mathbf{x}$
- Range of this function is linear subspace R and for any $\tilde{\mathbf{x}} \in R : \tilde{\mathbf{x}} = V\mathbf{y}$, for some vector y and orthonormal basis V
- Consider the minimum of $\|\mathbf{x} - V\mathbf{y}\|_2^2$. A bit of algebra and calculus shows that it is $\mathbf{y} = V^\top \mathbf{x}$ (because V is orthonormal).
- Therefore: $VV^\top \mathbf{x} = \arg\min_{\tilde{\mathbf{x}} \in R} \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2$
- And: $\sum_{i=1}^m \|\mathbf{x}_i - UW\mathbf{x}_i\|_2^2 \geq \sum_{i=1}^m \|\mathbf{x}_i - VV^\top \mathbf{x}_i\|_2^2$

How Do We Find $\mathbf{u}_1, \ldots, \mathbf{u}_n$?

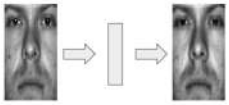
- Updated formal problem statement:

$$U = \arg\min_{U : U^\top U = I} \sum_{i=1}^m \|\mathbf{x}_i - UU^\top \mathbf{x}_i\|_2^2$$

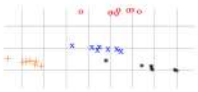
Face Reconstruction Example

Original data in $\mathbb{R}^{2500}$

Reconstruction via $\mathbb{R}^{10}$



Reduction to $\mathbb{R}^2$

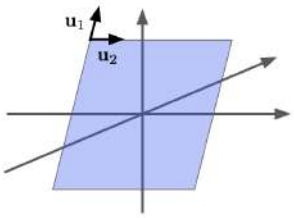


Images: Understanding Machine Learning, Shalev-Shwartz and Ben-David

Takeaway

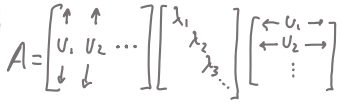
We've simplified the search for W and U to a search for the best linear subspace spanned by the orthonormal vectors $\mathbf{u}_1, \ldots, \mathbf{u}_n$

Example:



Principal Component Analysis

- Construct the matrix $A = \sum_{i=1}^m \mathbf{x}_i \mathbf{x}_i^\top$



- Take eigendecomposition of A: $A = UDU^\top$
- Set $\mathbf{u}_1, \ldots, \mathbf{u}_n$ to the eigenvectors corresponding to largest eigenvalues

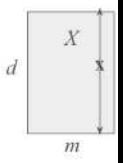
Center Data: s.t. $A = \sum_i$ (covariance) i.e. $A = \sum_{i=1}^m (x_i - \mu)(x_i - \mu)^\top$

Otherwise $u_1 \propto \mu$ (mean)

Transformers: Dominant NN-architecture

PCA in High Dimensions

- What happens if $d > m$?



- $A = \sum_{i=1}^m \mathbf{x}_i \mathbf{x}_i^\top = X^\top X$
- Let $B = XX^\top$ with eigenvectors $\mathbf{u}'_1, \ldots, \mathbf{u}'_m$
- Then $\frac{X^\top \mathbf{u}'}{\|X^\top \mathbf{u}'\|_2}$ is eigenvector of A with same eigenvalue
- Only $\mathcal{O}(m^3 + m^2 d)$ running time vs. $\mathcal{O}(d^3 + md^2)$!
- Also, B is a matrix of dot products of the data… kernel PCA!