[HMM]

Evaluation:
- Problem - Compute Probabilty of observation sequence given a model
- Solution - **Forward Algorithm** and **Viterbi Algorithm**

Decoding:
- Problem - Find state sequence which maximizes probability of observation sequence
- Solution - **Viterbi Algorithm**

Training:
- Problem - Adjust model parameters to maximize probability of observed sequences
- Solution - **Forward-Backward Algorithm**

Baum-Welch Reestimation:

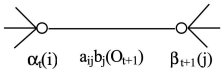
$$\bar{a}_{ij} = \frac{\text{expected number of trans from } S_i \text{ to } S_j}{\text{expected number of trans from } S_i} = \frac{\sum_{t=0}^{T-1} \xi_t(i,j)}{\sum_{t=0}^{T-1}\sum_{j=0}^{N} \xi_t(i,j)}$$

$$\bar{b}_j(k) = \frac{\text{expected number of times in state j with symbol k}}{\text{expected number of times in state j}} = \frac{\sum_{t \mid O_t = k} \sum_{i=0}^{N} \xi_t(i,j)}{\sum_{t=0}^{T-1}\sum_{i=0}^{N} \xi_t(i,j)}$$

$\xi_t(i,j) =$ **Probability of transiting from S_i to S_j at time t given O**

$$= P(\, q_t = S_i,\ q_{t+1} = S_j \mid O,\ \lambda\,)$$

$$= \frac{\alpha_t(i)\ a_{ij}\ b_j(O_{t+1})\ \beta_{t+1}(j)}{P(O \mid \lambda)}$$



Convergence of F-B Algorithm:

1. **Initialize $\lambda = (A,B)$**
2. **Compute α, β, and ξ**
3. **Estimate $\bar{\lambda} = (\bar{A}, \bar{B})$ from ξ**
4. **Replace λ with $\bar{\lambda}$**
5. **If not converged go to 2**

It can be shown that $P(O \mid \bar{\lambda}) > P(O \mid \lambda)$ unless $\bar{\lambda} = \lambda$

[Maximum Likelihood Estimation vs. Maximum a Posteriori]

$$\theta^{MLE} = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{N} p(x^{(i)} \mid \theta) \qquad \theta^{MAP} = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{N} p(x^{(i)} \mid \theta) p(\theta)$$

MLE: As much prob mass as possible to things we have observed
MAP: Hallucinate data
MLE of exponential with λ exp(-λx): MLE = N / Σ xi
MLE of Bernoulli with $\varphi_{MLE} = N_1 / (N_0 + N_1)$
MAP of Bernoulli-Beta(a,b): $\varphi_{MAP} = (N_1+a-1) / (N_0+N_1+a+b-2)$

[Generative vs. Discriminative]
Generative: models joint p(x, y), e.g. Naïve Bayes
Discriminative: models conditional p(y | x), e.g. Logistic Reg
If model assumption is correct, NB is a more efficient learner (requires fewer samples). Otherwise, LR has lower asymptotic error.

Naïve Bayes:
Parameters are decoupled → Closed form solution for MLE

Bernoulli Naïve Bayes:
Parameters are probabilities → Beta prior (usually) pushes probabilities away from zero / one extremes

Logistic Regression:
Parameters are coupled → No closed form solution – must use iterative optimization techniques instead

Logistic Regression:
Parameters are not probabilities → Gaussian prior encourages parameters to be close to zero

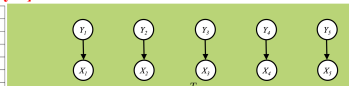
(effectively pushes the probabilities away from zero / one extremes)

Naïve Bayes:
Features $\boldsymbol{x}$ are assumed to be conditionally independent given y. (i.e. Naïve Bayes Assumption)

Logistic Regression:
No assumptions are made about the form of the features $\boldsymbol{x}$. They can be dependent and correlated in any fashion.

[Naïve Bayes]

A	B	C	K
1	0	1	1
1	1	1	1
0	1	1	0
1	1	0	0
1	0	1	0
0	0	0	1
0	0	0	1
0	0	1	0

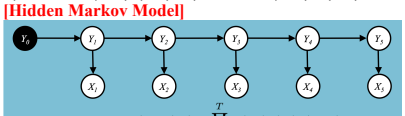


"Naïve Bayes": $P(\mathbf{X}, \mathbf{Y}) = \prod_{t=1}^{T} P(X_t \mid Y_t) p(Y_t)$

want P(K=1|A=1, B=1, C=0)?

$$P(K=1 \mid A=1, B=1, C=0) = \frac{P(A=1,B=1,C=0 \mid K=1)P(K=1)}{P(A=1,B=1,C=0 \mid K=1)P(K=1) + P(A=1,B=1,C=0 \mid K=0)\times P(K=0)}$$

[Hidden Markov Model]



HMM: $P(\mathbf{X}, \mathbf{Y} \mid Y_0) = \prod_{t=1}^{T} P(X_t \mid Y_t) p(Y_t \mid Y_{t-1})$

For random variables X_1, X_2, X_3, X_4:

$$P(X_1, X_2, X_3, X_4) = P(X_1 \mid X_2, X_3, X_4)\, P(X_2 \mid X_3, X_4)\, P(X_3 \mid X_4)\, P(X_4)$$

3 problems for HMM:
1. Evaluation: Compute the probability of a given sequence of observations. P(x) = Σ_y P (y, x)
2. Decode: Find the most-likely sequence of hidden states, given observations. y = argmax$_y$ P(y|x)
3. Marginal: Compute the marginal distribution for a hidden state, given a sequence of observations. P(y_t = k | x) = $\Sigma_{y:y_t=k}$ P(y|x)

Brutal force O(K^T), Verterbi & F-B O(TK^2)

Derivation of Forward Algorithm

Definition: $\alpha_t(k) \triangleq p(x_1, \ldots, x_t, y_t = k)$

Derivation:

$$\begin{aligned}
\alpha_T(END) &= p(x_1, \ldots, x_T, y_T = END) \\
&= p(x_1, \ldots, x_T \mid y_T)\, p(y_T) && \leftarrow \text{by def. of joint} \\
&= p(x_T \mid y_T)\, p(x_1, \ldots, x_{T-1} \mid y_T)\, p(y_T) && \leftarrow \text{by cond. indep. of HMM} \\
&= p(x_T \mid y_T)\, p(x_1, \ldots, x_{T-1}, y_T) && \leftarrow \text{by def. of joint} \\
&= p(x_T \mid y_T) \sum_{y_{T-1}} p(x_1, \ldots, x_{T-1}, y_{T-1}, y_T) && \leftarrow \text{by def. of marginal} \\
&= p(x_T \mid y_T) \sum_{y_{T-1}} p(x_1, \ldots, x_{T-1}, y_T \mid y_{T-1})\, p(y_{T-1}) && \leftarrow \text{by def. of joint} \\
&= p(x_T \mid y_T) \sum_{y_{T-1}} p(x_1, \ldots, x_{T-1} \mid y_{T-1})\, p(y_T \mid y_{T-1})\, p(y_{T-1}) && \leftarrow \text{by cond. indep. of HMM} \\
&= p(x_T \mid y_T) \sum_{y_{T-1}} p(x_1, \ldots, x_{T-1}, y_{T-1})\, p(y_T \mid y_{T-1}) && \leftarrow \text{by def. of joint} \\
&= p(x_T \mid y_T) \sum_{y_{T-1}} \alpha_{T-1}(y_{T-1})\, p(y_T \mid y_{T-1}) && \leftarrow \text{by def. of } \alpha_t(k)
\end{aligned}$$

Hereon using "y_T" as shorthand for "$y_T = END$"

Forward-Backward Algorithm

Define: $\alpha_t(k) \triangleq p(x_1, \ldots, x_t, y_t = k)$
$\beta_t(k) \triangleq p(x_{t+1}, \ldots, x_T \mid y_t = k)$

Assume $y_0 = START$, $y_{T+1} = END$

① Initialize $\alpha_0(START) = 1$, $\alpha_0(k) = 0\ \ \forall k \neq START$
$\beta_{T+1}(END) = 1$, $\beta_{T+1}(k) = 0\ \ \forall k \neq END$

② For $t = 1, \ldots, T$:
For $k = 1, \ldots, K$:
$\alpha_t(k) = p(x_t \mid y_t = k) \sum_{j=1}^{K} \alpha_{t-1}(j)\, p(y_t = k \mid y_{t-1} = j)$
(the alphas include the emission probabilities so we don't multiply them in separately)

③ For $t = T, \ldots, 1$:
For $k = 1, \ldots, K$:
$\beta_t(k) = \sum_{j=1}^{K} p(x_{t+1} \mid y_{t+1} = j)\, \beta_{t+1}(j)\, p(y_{t+1} = j \mid y_t = k)$

④ Compute $p(\vec{x}) = \alpha_{T+1}(END)$ [Evaluation]

⑤ Compute $p(y_t = k \mid \vec{x}) = \frac{\alpha_t(k)\, \beta_t(k)}{p(\vec{x})}$ [Marginals]

Viterbi Algorithm

Define: $\omega_t(k) \triangleq \max_{y_1, \ldots, y_{t-1}} p(x_1, \ldots, x_t, y_1, \ldots, y_{t-1}, y_t = k)$

"backpointers" $b_t(k) \triangleq \underset{y_1, \ldots, y_{t-1}}{\operatorname{argmax}} p(x_1, \ldots, x_t, y_1, \ldots, y_{t-1}, y_t = k)$

Assume $y_0 = START$

① Initialize $\omega_0(START) = 1$ $\omega_0(k) = 0\ \ \forall k \neq START$

② For $t = 1, \ldots, T$:
For $k = 1, \ldots, K$:
$\omega_t(k) = \max_{j \in \{1, \ldots, K\}} p(x_t \mid y_t = k)\, \omega_{t-1}(j)\, p(y_t = k \mid y_{t-1} = j)$
$b_t(k) = \underset{j \in \{1, \ldots, K\}}{\operatorname{argmax}} p(x_t \mid y_t = k)\, \omega_{t-1}(j)\, p(y_t = k \mid y_{t-1} = j)$

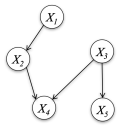
③ Compute Most Probable Assignment [Decoding]
$\hat{y}_T = b_{T+1}(END)$
For $t = T-1, \ldots, 1$
$\hat{y}_t = b_{t+1}(\hat{y}_{t+1})$
follow the "backpointers"

Short Comings:
- HMM models capture dependences between each state and only its corresponding observation
- Mismatch between learning objective function P(X,Y) and prediction objective function P(Y|X)

Minimum Bayes Risk Decoding: (0-1 loss == verterbi)

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underset{\hat{\boldsymbol{y}}}{\operatorname{argmin}}\ \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})] = \underset{\hat{\boldsymbol{y}}}{\operatorname{argmin}} \sum_{\boldsymbol{y}} p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \ell(\hat{\boldsymbol{y}}, \boldsymbol{y})$$

[Direct Graphical Models (Bayes Net)]



$$p(X_1, X_2, X_3, X_4, X_5) = p(X_5 \mid X_3)\, p(X_4 \mid X_2, X_3)\, p(X_3)\, p(X_2 \mid X_1)\, p(X_1)$$

In order for a Bayesian network to model a probability distribution, the following must be true:

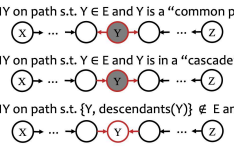
Each variable is conditionally independent of all its non-descendants in the graph given the value of all its parents.

D-Separation:

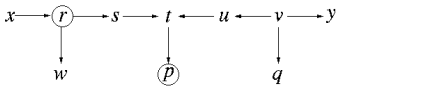
If variables X and Z are d-separated given a **set** of variables E
Then X and Z are conditionally independent given the **set** E

Definition #1:
Variables X and Z are d-separated given a **set** of evidence variables E iff every path from X to Z is "blocked".

A path is "blocked" whenever:
1. ∃Y on path s.t. Y ∈ E and Y is a "common parent"
2. ∃Y on path s.t. Y ∈ E and Y is in a "cascade"
3. ∃Y on path s.t. {Y, descendants(Y)} ∉ E and Y is in a "v-structure"



Rule 3: If a collider is a member of the conditioning set Z, or has a descendant in Z, then it no longer blocks any path that traces this collider.



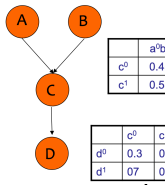
Quantitative Specification

Example: Conditional probability tables (CPTs) for discrete random variables

a^0	0.75
a^1	0.25

b^0	0.33
b^1	0.67

P(a,b,c.d) = P(a)P(b)P(c|a,b)P(d|c)



	a^0b^0	a^0b^1	a^1b^0	a^1b^1
c^0	0.45	1	0.9	0.7
c^1	0.55	0	0.1	0.3

	c^0	c^1
d^0	0.3	0.5
d^1	07	0.5

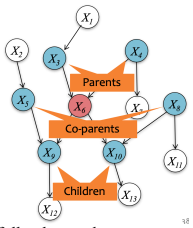
Markov Blanket

Def: the **co-parents** of a node are the parents of its children

Def: the **Markov Blanket** of a node is the set containing the node's parents, children, and co-parents.

Theorem: a node is **conditionally independent** of every other node in the graph given its **Markov blanket**

Example: The Markov Blanket of X_6 is $\{X_3, X_4, X_5, X_8, X_9, X_{10}\}$



Learning a Bayes Net: Learning this fully observed Bayesian Network is equivalent to learning five (small / simple) independent networks from the same data

[Reinforcement Learning]

Fixed point iteration:

$$J(x) = \frac{x^3}{3} + \frac{3}{2}x^2 + 2x$$

$$\frac{dJ(x)}{dx} = f(x) = x^2 - 3x + 2 = 0$$

$$\Rightarrow x = \frac{x^2 + 2}{3} = g(x)$$

$$x \leftarrow \frac{x^2 + 2}{3}$$

1. Given objective function:
2. Compute derivative, set to zero (call this function f).
3. Rearrange the equation s.t. one of parameters appears on the LHS.
4. Initialize the parameters.
5. For i in $\{1, \ldots, K\}$, update each parameter and increment t:
6. Repeat #5 until convergence

Value iteration and Policy iteration are two more classic approaches to this problem. But essentially both are dynamic programming.

Q-learning is a more recent approaches to this problem. Essentially it is a temporal-difference method.

Algorithm 1 Value Iteration

```
procedure VALUEITERATION(R(s,a) reward function, p(·|s,a) transition probabilities)
    Initialize value function V(s) = 0 or randomly
    while not converged do
        for s ∈ S do
            for a ∈ A do
                Q(s,a) = R(s,a) + γ Σ_{s'∈S} p(s'|s,a) V(s')
            V(s) = max_a Q(s,a)
    Let π(s) = argmax_a Q(s,a), ∀s
    return π
```

O(|A| $|S|^2$) calculations per iteration, O(|A|$^{|S|}$) steps to converge (number of all possible policies)

Algorithm 1 Policy Iteration

```
procedure POLICYITERATION(R(s,a) reward function, p(·|s,a) transition probabilities)
    Initialize policy π randomly
    while not converged do
        Solve Bellman equations for fixed policy π
            V^π(s) = R(s,π(s)) + γ Σ_{s'∈S} p(s'|s,π(s)) V^π(s'),  ∀s
        Improve policy π using new value function
            π(s) = argmax_a R(s,a) + γ Σ_{s'∈S} p(s'|s,a) V^π(s')
    return π
```

$O(|A| \ |S|^2 + |S|^3)$ calculations per iteration, converge faster.

Algorithm Q-Learning

$V^*(s) = \max_a Q^*(s,a) \quad \pi^*(s) = \arg\max_a Q^*(s,a)$

$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q^*(s',a')$

1. Initialize $Q(s,a) = 0 \ \ \forall s, a$
2. Do forever
   1. Select action a and execute
   2. receive reward $r = R(s,a)$

[Deterministic]

   3. observe new state $s' = \delta(s,a)$
   4. update table entry $Q(s,a) = r + \gamma \max_{a'} Q(s',a')$

[Nondeterministic]

   3. observe new state $s' \sim p(\cdot|s,a)$
   4. update table entry $Q(s,a) = (1-\alpha_n)Q(s,a) + \alpha_n(r + \gamma \max_{a'} Q(s',a'))$, where $\alpha_n = 1/(1+K_n(s,a))$

$\alpha\_n = \frac{1}{1+\#visits\ at\ time\ stamp\ n\ at\ a\ (s,a)\ pair}$

Epsilon Greedy:

w.p. $1-\epsilon$, select a = $\max_{a'} Q(s, a')$,

w.p. ϵ, select a = random.

[Support Vector Machine]

Maximize "margin"

(There exists a **w** and **b,**) s.t. $y^{(i)}(\mathbf{w}\mathbf{x} + \mathbf{b}) >= 1$

margin/width = $d_+ + d_-$

$d_+ = \mathbf{w}x_+ / \|\mathbf{w}\|_2 + \mathbf{b}$, $d_- = -\mathbf{w}x_- / \|\mathbf{w}\|_2 - \mathbf{b}$

Constraint is tight for x_+ and x_-, which means:

$\mathbf{w}x_+ + \mathbf{b} = 1$, $\mathbf{w}x_- + \mathbf{b} = -1$,

minimize margin/width = $2 / \|\mathbf{w}\|_2$

maximize $\|\mathbf{w}\|_2 / 2$

Outliers doesn't matter for SVM but does for LR

Hard-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$
$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \quad \forall i = 1, \ldots, N$$

Hard-margin SVM (Lagrangian Dual)

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y^{(j)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$$
$$\text{s.t. } \alpha_i \geq 0, \quad \forall i = 1, \ldots, N$$
$$\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

Hard-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$
$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \quad \forall i = 1, \ldots, N$$

Hard-margin SVM (Lagrangian Dual)

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$$
$$\text{s.t. } \alpha_i \geq 0, \quad \forall i = 1, \ldots, N$$
$$\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

Soft-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\left(\sum_{i=1}^{N} e_i\right)$$
$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - e_i, \quad \forall i = 1, \ldots, N$$
$$e_i \geq 0, \quad \forall i = 1, \ldots, N$$

Soft-margin SVM (Lagrangian Dual)

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$$
$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad \forall i = 1, \ldots, N$$
$$\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

1. ***one-vs-rest:***
   - build K binary classifiers
   - train the kth classifier to predict whether an instance has label k or something else
   - predict the class with largest score
2. ***one-vs-one:***
   - build (K choose 2) binary classifiers
   - train one classifier for distinguishing between each pair of labels
   - predict the class with the most "votes" from any given classifier

$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\vec{w}^T\vec{w} - \sum_{i=1}^{N} \alpha_i [y^{(i)}(\vec{w}^T\vec{x}^{(i)} + b) - 1]$

Primal: $\min_{\vec{w},b} \max_{\vec{\alpha}} L(\vec{w}, b, \vec{\alpha})$ equivalent by Slater's condition Dual: $\max_{\vec{\alpha}} \min_{\vec{w},b} L(\vec{w}, b, \vec{\alpha})$

Lagrangian Multipliers

$$\min_w \ f(w)$$
$$\text{s.t.} \quad g_i(w) \leq 0, i = 1, \ldots, k$$
$$h_i(w) = 0, i = 1, \ldots, l$$

This is also called the primal problem. We introduce new variables $\alpha_i \geq 0$ and β_i (no need to be non-negative) called Lagrange multipliers and study the generalized Lagrangian defined by

$$L(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

Solve $\frac{\partial L(w,\alpha,\beta)}{\partial w} = 0, \frac{\partial L(w,\alpha,\beta)}{\partial \alpha} = 0, \frac{\partial L(w,\alpha,\beta)}{\partial \beta} = 0.$

s.t. $\lambda \geq 0, g(w) \leq 0$ (*or* C).

[Matrix Factorization]

SGD for UMF:

While not converged:

① Sample (i,j) from $\mathbb{Z}$ uniformly at random

② Compute $e_{ij} = r_{ij} - \vec{v}_i^T\vec{v}_j$

③ Update

$\vec{v}_i \leftarrow \vec{v}_i - \gamma \nabla_{\vec{v}_i} J_{ij}(U,V)$

$\vec{v}_j \leftarrow \vec{v}_j - \gamma \nabla_{\vec{v}_j} J_{ij}(U,V)$ W/ Regularization

$J_{ij}(U,V) = \frac{1}{2}(r_{ij} - \vec{v}_i^T\vec{v}_j)^2 + \lambda(\|v_i\|_2^2 + \|v_j\|_2^2)$

$\nabla_{\vec{v}_i} J_{ij}(U,V) = -e_{ij}\vec{v}_j + \lambda\vec{v}_i$

$\nabla_{\vec{v}_j} J_{ij}(U,V) = -e_{ij}\vec{v}_i + \lambda\vec{v}_j$

where $e_{ij} = r_{ij} - \vec{v}_i^T\vec{v}_j$

[Principle Component Analysis]

Assume data is 0-centered and variance along each axis is 1.

Equivalence of Maximizing Variance and Minimizing Reconstruction Error

Claim: Minimizing the reconstruction error is equivalent to maximizing the variance.

Proof: First, note that:

$$\|\mathbf{x}^{(i)} - (\mathbf{v}^T\mathbf{x}^{(i)})\mathbf{v}\|^2 = \|\mathbf{x}^{(i)}\|^2 - (\mathbf{v}^T\mathbf{x}^{(i)})^2 \quad (1)$$

since $\mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|^2 = 1$.

Substituting into the minimization problem, and removing the extraneous terms, we obtain the maximization problem.

$$\mathbf{v}^* = \underset{\mathbf{v}:\|\mathbf{v}\|^2=1}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}^{(i)} - (\mathbf{v}^T\mathbf{x}^{(i)})\mathbf{v}\|^2 \quad (2)$$

$$= \underset{\mathbf{v}:\|\mathbf{v}\|^2=1}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}^{(i)}\|^2 - (\mathbf{v}^T\mathbf{x}^{(i)})^2 \quad (3)$$

$$= \underset{\mathbf{v}:\|\mathbf{v}\|^2=1}{\operatorname{argmax}} \frac{1}{N}\sum_{i=1}^{N} (\mathbf{v}^T\mathbf{x}^{(i)})^2 \quad (4)$$

$$(5)$$

V is first K cols of V in SVD of X = USV^T

[Ensemble Methods]

Weighted Majority:

Suppose we have a pool of T binary classifiers $\mathcal{A} = \{h_1, \ldots, h_T\}$ where $h_t : \mathbb{R}^M \to \{+1, -1\}$. Let α_t be the weight for classifier h_t.

Algorithm 1 Weighted Majority Algorithm

procedure WEIGHTEDMAJORITY($\mathcal{A}, \beta$)
  Initialize classifier weights $\alpha_t = 1, \ \forall t \in \{1, \ldots, T\}$
  for each training example $(\mathbf{x}, y)$ **do**
    Predict majority vote class (splitting ties randomly)

$$\hat{h}(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

    if a mistake is made $\hat{h}(x) \neq y$ **then**
      for each classifier $t \in \{1, \ldots, T\}$ **do**
        If $h_t(x) \neq y$, then $\alpha_t \leftarrow \beta\alpha_t$

Theorem 0.1 (Littlestone & Warmuth, 1994). *If the Weighted Majority Algorithm is applied to a pool $\mathcal{A}$ of classifiers, and if each algorithm makes at most m mistakes on the sequence of examples, then the total number of mistakes is upper bounded by $2.4(\log|\mathcal{A}| + m)$.*

AdaBoost

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$.

For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i].$$

- Choose $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$
$$= \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Weak learners. Adaboost never overfits.

[K-Nearest Neighbor]

2.1 Euclidean distance: $g(\boldsymbol{u}, \boldsymbol{v}) = \sqrt{\sum_{m=1}^{M}(u_m - v_m)^2}$

Manhattan distance: $g(\boldsymbol{u}, \boldsymbol{v}) = \sqrt{\sum_{m=1}^{M}|u_m - v_m|}$

2.2 Questions on KNN

1. How to handle even values of k?
   1. keep adding points 2. distance-weighted 3. remove furthest point...
2. What is the inductive bias of KNN?
   1. Similar points should have similar labels
   2. All dimensions are created equally
3. Computational Efficiency of KNN?
   - Suppose we have N training examples, and each one has M features.
   - When training, Naive (k = 1) is $O(1)$, k-d Tree is $O(MN \log N)$
   - When testing, Naive (k = 1) is $O(MN)$, k-d Tree is $O(2^M \log N)$
4. Theoretical Guarantees of KNN?
   - $error_{true}(h) < 2 \times$ Bayes error rate (Cover & Hart, 1967)

2.3 Cross Validation

Leave out different folds as validation sets, and get different validation error curves. Then we can average the curves and get a smoother validation error curve, and finally choose model with lowest validation error.

[Perceptron]

3.1 Definition

h(x) = sign($w^Tx + b$) where sign(x) = 1 if x >= 0 else -1

Note that 0 = $w^Tx + b$ is orthogonal to w

3.2 (Online) Perceptron Algorithm

Initialize parameters: weights $\boldsymbol{w} = [w_1, w_2, \cdots, w_M]^T = \mathbf{0}$, bias $b = 0$.

for i = 1, 2, 3, ...

1. Receive next example $(\boldsymbol{x}^{(i)}, y^{(i)})$
2. Predict $\hat{y} = \text{sign}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b)$
3. Switch on cases
   - If false negative, i.e. $\hat{y} \neq y^{(i)} = 1$, $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}^{(i)}, b = b + 1$
   - If false positive, i.e. $\hat{y} \neq y^{(i)} = -1$, $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{x}^{(i)}, b = b - 1$

Note: the hypothesis space of perceptron algorithm is all possible linear decision boundaries: $\mathcal{H} = \{h(\cdot) \mid \exists \boldsymbol{w} \in \mathbb{R}^M, \ b \in R, \text{ s.t. } h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b\}$

Why this algorithm works?

- for $|\boldsymbol{w}|_2 = 1$, $\boldsymbol{w}^T\boldsymbol{x}$ is the length of vector project of $\boldsymbol{x}$ onto $\boldsymbol{w}$
- wrongly predicted $\boldsymbol{x}$ corrects $\boldsymbol{w}$
- b shifts the zero points where $\boldsymbol{w}$ starts

Note: the inductive bias of perceptron algorithm is that

- decision boundary should be linear
- prefers to correct most recent mistakes

3.3 Batch Perceptron Algorithm

procedure PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{\{1\}}, y^{\{1\}}), \ldots, (\mathbf{x}^{\{N\}}, y^{\{N\}})\}$)
  $\boldsymbol{\theta} \leftarrow \mathbf{0}$ ▷ Initialize parameters
  while not converged **do**
    for $i \in \{1, 2, \ldots, N\}$ **do** ▷ For each example
      $\hat{y} \leftarrow \text{sign}(\boldsymbol{\theta}^T\mathbf{x}^{(i)})$ ▷ Predict
      if $\hat{y} \neq y^{(i)}$ **then** ▷ If mistake
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)}\mathbf{x}^{(i)}$ ▷ Update parameters
  return $\boldsymbol{\theta}$

Note: It cannot converge on not linearly separable data

3.4 Perceptron Mistake Bound: With margin γ and all points inside a ball of radius R, it makes <= $(R / \gamma)^2$ mistakes

[Linear Regression and Optimization]

4.1 Linear Regression as Function Approximation

(1) Assume D is generated as $\boldsymbol{x}^{(i)} \sim p^*(\boldsymbol{x})$, $y^{(i)} \sim c^*(\boldsymbol{x}^{(i)})$, where p^* and c^* are unknown

(2) Choose $\mathcal{H} = \{h_{\boldsymbol{\theta}} | h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T\boldsymbol{x}, \ \boldsymbol{\theta} \in \mathbb{R}^{M+1}, x_0 = 1\}$

(3) Choose a objective function with a goal of minimizing mean squared

$$\mathcal{J}_D(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} e_i^2 = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - (\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b))^2$$

(4) Solve optimization problem $\hat{\boldsymbol{\theta}} = \text{argmin}_{\boldsymbol{\theta}}\mathcal{J}_D(\boldsymbol{\theta})$

(5) Predict, given next $\boldsymbol{x}$, $y = h_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}) = \hat{\boldsymbol{\theta}}^T\boldsymbol{x}$

4.2 Convex Function: $f(t\,x_1 + (1-t)\,x_2) <= t\,f(x_1) + (1-t)f(x_2)$

4.3 Gradient Descent:

procedure GD($\mathcal{D}, \boldsymbol{\theta}^{\{0\}}$)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{\{0\}}$
  while not converged **do**
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$
  return $\boldsymbol{\theta}$

4.4 GD for Linear Regression:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{n}\frac{1}{2}(y^{(i)} - \boldsymbol{\theta}^T\boldsymbol{x}^{(i)})^2$$
$$\frac{\partial\mathcal{J}_i(\boldsymbol{\theta})}{\partial\theta_j} = -(y^{(i)} - \boldsymbol{\theta}^T\boldsymbol{x}^{(i)})\boldsymbol{x}_m^{(i)}$$
$$\nabla\mathcal{J}_i(\boldsymbol{\theta}) = -(y^{(i)} - \boldsymbol{\theta}^T\boldsymbol{x}^{(i)})\boldsymbol{x}^{(i)}$$
$$\nabla\mathcal{J}(\boldsymbol{\theta}) = -\sum_{i=1}^{n}(y^{(i)} - \boldsymbol{\theta}^T\boldsymbol{x}^{(i)})\boldsymbol{x}^{(i)}$$

4.5 Stochastic Gradient Descent

procedure SGD($\mathcal{D}, \boldsymbol{\theta}^{\{0\}}$)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{\{0\}}$
  while not converged **do**
    for $i \in$ shuffle$\{1, 2, \ldots, N\}$ **do**
      for $k \in \{1, 2, \ldots, K\}$ **do**
        $\theta_k \leftarrow \theta_k - \lambda\frac{d}{d\theta_k}J^{\{i\}}(\boldsymbol{\theta})$
  return $\boldsymbol{\theta}$

Note: Expectation of a randomly chosen $\nabla J^{(i)}(\theta)$ is $\nabla J(\theta)$

Note: Steps to converge: GD O(ln 1/error), SGD O(1/error).

Note: Computation per iteration: GD O(NM), SGD O(M)

[Logistic Regression]

5.1 Logistic Regression and its SGD

Learning:

$\boldsymbol{\theta}^* = \text{argmin}\ J(\boldsymbol{\theta})$

Prediction:

$\hat{y} = \text{argmax}\ p_{\boldsymbol{\theta}}(y|\mathbf{x})$

procedure SGD($\mathcal{D}, \boldsymbol{\theta}^{\{0\}}$)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{\{0\}}$
  while not converged **do**
    for $i \in$ shuffle$\{1, 2, \ldots, N\}$ **do**
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda(y^{(i)} - \rho^{(i)})\mathbf{x}^{(i)}$
      where $\rho^{(i)} := 1/(1 + \exp(-\boldsymbol{\theta}^T\mathbf{x}))$
  return $\boldsymbol{\theta}$

5.2 Multinomial

Regression	Binomial	Multinomial (n>2)
$\hat{y}$	$\hat{y} = \frac{1}{1+e^{-\theta X}}$	$\hat{y} = \frac{e^{\theta_y X}}{\sum e^{\theta X}}$
cross entropy	$J(\theta) = -\frac{1}{n}\sum_i^n [y_i log(\hat{y}_i) + (1-y_i) log(1-\hat{y}_i)]$	$J(\theta) = -\sum_i y_i ln(\hat{y}_i)$
forward pass	$[NX1] = [NXD][DX1]$	$[NXC] = [NXD][DXC]$
response	$y = 0$ or 1	y = one-hot-encoded

Backpropagation:

Logistic Regression:

Forward	Backward
$J = y^* \log y + (1-y^*)\log(1-y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1-y^*)}{y-1}$
$y = \frac{1}{1+\exp(-a)}$	$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a)+1)^2}$
$a = \sum_{j=0}^{D}\theta_j x_j$	$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \frac{da}{d\theta_j} = x_j$ $\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \frac{da}{dx_j} = \theta_j$

Neural Net:

Case 2:	Forward	Backward
Loss	$J = y^* \log y + (1-y^*)\log(1-y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1-y^*)}{y-1}$
Sigmoid	$y = \frac{1}{1+\exp(-b)}$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b)+1)^2}$
Linear	$b = \sum_{j=0}^{D}\beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$ $\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$
Sigmoid	$z_j = \frac{1}{1+\exp(-a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j)+1)^2}$
Linear	$a_j = \sum_{i=0}^{M}\alpha_{ji}x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \sum_{j=0}^{D}\frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \alpha_{ji}$