

Machine Learning

Liyao Tang

October 9, 2018

Contents

1	Math	1
1.1	Convolution	1
1.2	Normal Distribution	2
2	Statistical Machine Learning	4
2.1	Supervised Learning	4
2.2	Linear Regression	4
2.3	Bayesian Regression	6
2.4	Logistic Regression (Classification)	8
2.5	Diagnose Machine Learning	13
2.5.1	Evaluating Hypothesis	13
2.5.2	Bias / Variance	14
2.5.3	Error Analysis	15
2.5.4	Skewed classes	15
2.6	Latent Variable Analysis	15
2.6.1	Principal Component Analysis (PCA)	15
2.6.2	Independent Component Analysis (ICA)	19
2.6.3	t-SNE	19
2.6.4	Anomaly Detection	19
2.6.5	Recommender System	20
2.7	Large Scale Machine Learning	22
2.7.1	Gradient Descent with Large Dataset	22
2.7.2	Online Learning	23
2.7.3	Map-reduce	23
2.8	Building Machine Learning System	24
2.8.1	Pipeline	24
2.8.2	Getting More Data	24
2.8.3	Ceiling Analysis	24

List of Figures

List of Tables

Chapter 1

Math

1.1 Convolution

- Definition

- $f * g(z) = \int_{\mathbb{R}} f(x)g(z-x)dx$, where $f(x), g(x)$ are functions in \mathbb{R}

- Statistical Meaning

- Notation

- X, Y : independent random variables, with pdf's given by f and g
 - $Z = X + Y$, with pdf given by $h(z)$:

- $\Rightarrow h(z) = f * g(z)$

- derivation

$$\begin{aligned} H(z) &= P(Z < z) = P(X + Y < z) \\ &= \int_x P(X = x)P(X + Y < z | X = x)dx \\ &= \int_x f(x)P(Y < z - x)dx \\ &= \int_x f(x)G_Y(z - x)dx \end{aligned}$$

$$\begin{aligned} \Rightarrow h(x) &= \frac{d}{dz}H(z) = \frac{d}{dz} \int_x f(x)G_Y(z - x)dx \\ &= \int_x f(x) \frac{dG_Y(z - x)}{dz} dx \\ &= \int_x f(x)g(z - x)dx \\ &= f * g(z) \end{aligned}$$

1.2 Normal Distribution

- d -dimensional Gaussian

- Variable

- mean: $\mu \in \mathbb{R}^d$

- covariance matrix: $\Sigma_{d \times d}$

- PDF

- $g_d(x|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$,
noted as $X \sim \mathcal{N}_d(x|\mu, \Sigma)$

- Convolution of Gaussian

- Integral of Gaussians $\int G_1 G_2 dx$

- $G_1 \sim \mathcal{N}_d(x|a, A), G_2 \sim \mathcal{N}_d(x|b, B)$

$$\begin{aligned}
& \Rightarrow \int \mathcal{N}_d(x|a, A) \mathcal{N}_d(x|b, B) dx \\
&= \int \frac{1}{(2\pi)^{d/2}|A|^{1/2}} e^{-\frac{1}{2}(x-a)^T A^{-1}(x-a)} \frac{1}{(2\pi)^{d/2}|B|^{1/2}} e^{-\frac{1}{2}(x-b)^T B^{-1}(x-b)} dx \\
&= \int \frac{1}{(2\pi)^{d/2}|A|^{1/2}} \frac{1}{(2\pi)^{d/2}|B|^{1/2}} e^{-\frac{1}{2}[(x-a)^T A^{-1}(x-a) + (x-b)^T B^{-1}(x-b)]} \\
&= \int \frac{1}{(2\pi)^{d/2}|A|^{1/2}} \frac{1}{(2\pi)^{d/2}|B|^{1/2}} e^{-\frac{1}{2}[(x-c)^T (A^{-1}+B^{-1})(x-c) + (a-b)^T C(a-b)]}, \\
&\quad \text{where } c = (A^{-1} + B^{-1})^{-1}(A^{-1}a + B^{-1}b), C = A^{-1}(A^{-1} + B^{-1})^{-1}B^{-1} = (A + B)^{-1} \\
&= \frac{|(A^{-1} + B^{-1})^{-1}|^{1/2}}{(2\pi)^{d/2}|A|^{1/2}|B|^{1/2}} e^{-\frac{1}{2}(a-b)^T C(a-b)} \int \frac{1}{(2\pi)^{d/2}|(A^{-1} + B^{-1})^{-1}|^{1/2}} e^{-\frac{1}{2}(x-c)^T (A^{-1}+B^{-1})(x-c)} dx \\
&= \frac{|(A^{-1} + B^{-1})^{-1}|^{1/2}}{(2\pi)^{d/2}|A|^{1/2}|B|^{1/2}} e^{-\frac{1}{2}(a-b)^T C(a-b)} \\
&= \frac{1}{(2\pi)^{d/2}(|A||B||A^{-1} + B^{-1}|)^{1/2}} e^{-\frac{1}{2}(a-b)^T (A+B)^{-1}(a-b)} \\
&= \frac{1}{(2\pi)^{d/2}|ABA^{-1} + ABB^{-1}|^{1/2}} e^{-\frac{1}{2}(a-b)^T (A+B)^{-1}(a-b)} \\
&= \frac{1}{(2\pi)^{d/2}|ABA^{-1} + A|^{1/2}} e^{-\frac{1}{2}(a-b)^T (A+B)^{-1}(a-b)} \\
&= \frac{1}{(2\pi)^{d/2}|A(B+A)A^{-1}|^{1/2}} e^{-\frac{1}{2}(a-b)^T (A+B)^{-1}(a-b)} \\
&= \frac{1}{(2\pi)^{d/2}|A+B|^{1/2}} e^{-\frac{1}{2}(a-b)^T (A+B)^{-1}(a-b)}
\end{aligned}$$

- \Rightarrow Convolution of Gaussians $G_1 * G_2$

$$\blacksquare G_1 \sim \mathcal{N}_d(a, A), G_2 \sim \mathcal{N}_d(b, B)$$

$$\begin{aligned}
G_1 * G_2(z) &= \int G_1(x) G_2(z - x) dx \\
&= \int \mathcal{N}_d(x|a, A) \mathcal{N}_d(z - x|b, B) dx \\
&= \int \mathcal{N}_d(x|a, A) \cdot \frac{1}{(2\pi)^{d/2} |B|^{1/2}} e^{-\frac{1}{2}(z-x-b)^T B^{-1}(z-x-b)} dx \\
&= \int \mathcal{N}_d(x|a, A) \mathcal{N}_d(x|z - b, B) dx \\
&= \frac{1}{(2\pi)^{d/2} |A + B|^{1/2}} e^{-\frac{1}{2}(z-(a+b))^T (A+B)^{-1}(z-(a+b))} \\
&= \mathcal{N}_d(z|\mathbf{a} + \mathbf{b}, \mathbf{A} + \mathbf{B})
\end{aligned}$$

Chapter 2

Statistical Machine Learning

2.1 Supervised Learning

- Feature normalization: $\forall x_{ij} \in X, x_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$, $X : [instance][feature]$, without $[1...1]^T$ in 1st column $X = [x_1, x_2, \dots, x_m]$, m instances in total
- Regularization: add penalty for θ being large into cost function
- $J(\theta) = \dots + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$, θ_0 **shouldn't be penalized**
- Matlab/Octave tips:
 - Unrolling parameters: `deltaVector = [D1(:) ; D2(:)] // matrix`
`D1,D2 = column vector`
 - Reshape: `D1 = reshape(deltaVector(1, m*n), m, n) // column vector`
 $\Rightarrow m*n$ matrix

2.2 Linear Regression

- Assumption:
 - $y = h_{\theta}^*(x) + \epsilon$, where $h_{\theta}^*(x)$ is the theoretical (perfect) hypothesis, ϵ is gaussian noise
- General Hypothesis Function: $h_{\theta}(x) = \phi(x)\theta$, where $\phi(x)$ is basis function
- Log Likelihood leads to least square cost function $J(\theta)$
 - $P(y|x, \theta, \beta) = \mathcal{N}(x|h_{\theta}(x), \beta^{-1})$ - likelihood

- based on assumption: $y = h_\theta^*(x) + \epsilon$, where $h_\theta^*(x)$ is the theoretical (perfect) hypothesis, ϵ is gaussian noise
 - $P(\mathbf{y}|X, \theta, \beta) = \prod_{i=1}^m \mathcal{N}(x^i | h_\theta(x^i), \beta^{-1})$
 - $\ln P(\mathbf{y}|X, \theta, \beta) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta \frac{1}{2} \sum_{i=1}^m (h_\theta(x^i) - y^i)^2$
- Log Posterior leads to regularization
 - Maximizing the likelihood function \Rightarrow (often) excessively complex models & over-fitting
 - Regularization term comes from the Prior:
 - assume Prior $p(\theta) = \mathcal{N}(\theta|0, \alpha^{-1}I)$, so that Posterior & Prior are of the same distribution to maximize log Posterior :

$$\Rightarrow \ln p(\theta|X) \propto -\frac{\beta}{2} \sum_{i=1}^n (y^i - h_\theta(x))^2 - \frac{\alpha}{2} \theta^T \theta + \text{const}$$
 - If $\alpha \rightarrow 0$ (Prior is most useless), maximise Posterior is equivalent to maximizing likelihood
 - Maximize Posterior \Leftrightarrow Minimize cost function with regularization, where $\lambda = \alpha/\beta$
- Predictive Distribution: $p(y|x, X, Y)$
 - $p(y|x, X, Y) = \int p(y, \theta|x, X, Y) d\theta = \int p(y|\theta, x, X, Y) p(\theta|x, X, Y) d\theta$
 - $p(y|\theta, x, X, Y) = p(y|\theta, x) = \mathcal{N}(y|h(x, \theta), \beta^{-1})$
 based on assumption: $y = y(x, \theta) + \epsilon$, where ϵ is Gaussian noise
 $p(\theta|x, X, Y) = p(\theta|X, Y) = \text{posterior}$
 - $\Rightarrow p(y|x, X, Y) = \int p(y|\theta, x) p(\theta|X, Y) d\theta$
 - Expected Lost = $(\text{bias})^2 + \text{variance} + \text{noise}$
- Notation:
 - $t = y(x, w) + \epsilon$, where ϵ is Gaussian noise
 - \hat{y} is prediction function to approximate $y = y(x, w)$
- Procedure:
 - $\mathbb{E}[(t - \hat{y})^2] = \mathbb{E}[t^2 - 2t\hat{y} + \hat{y}^2]$
 $= \mathbb{E}[t^2] + \mathbb{E}[\hat{y}^2] - \mathbb{E}[2t\hat{y}]$
 $= \text{Var}[t] + \mathbb{E}[t]^2 + \text{Var}[\hat{y}] + \mathbb{E}[\hat{y}]^2 - 2y\mathbb{E}[\hat{y}]$
 $= \text{Var}[t] + \text{Var}[\hat{y}] + (y^2 - 2y\mathbb{E}[\hat{y}] + \mathbb{E}[\hat{y}]^2)$

$$= \text{Var}[t] + \text{Var}[\hat{y}] + (y - \mathbb{E}[\hat{y}])^2$$

$$= \text{Var}[t] + \text{Var}[\hat{y}] + \mathbb{E}[t - \hat{y}]^2$$

$$= \sigma^2 + \text{Var}[\hat{y}] + \text{Bias}[\hat{y}]^2$$

where $\sigma^2 = \text{Var}[\epsilon]$ is the noise

$$(\text{formula used: } \text{Var}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 \Leftrightarrow \mathbb{E}[x^2] = \text{Var}[x] + \mathbb{E}[x]^2)$$

- Matrix inverse can be evil & avoid inverse operation:

$$A = U\Lambda U^T, \text{ where } \Lambda \text{ is diagonal matrix}$$

$$\Rightarrow A^{-1} = U\Lambda^{-1}U^T$$

but number on the diagonal line of Λ can be small \Rightarrow maybe 0 depending on accuracy of computer

2.3 Bayesian Regression

- Assumption:

- $t = y(x, w) + \epsilon$, where ϵ is Gaussian noise; $y(x, w)$ approximated by $\phi(x)w$

- Bayesian view:

- Gaussian Prior : $p(w) = \mathcal{N}(w|m_0, S_0)$

Reason: to be conjugate

- Likelihood : $p(\mathbf{t}|w) = \prod_{n=1}^N \mathcal{N}(t_n|w^T \phi(x_n), \beta^{-1}) = \mathcal{N}(\mathbf{t}|\Phi w, \beta^{-1}I)$

- \Rightarrow Posterior : $p(w|\mathbf{t}) = \mathcal{N}(w|m_N, S_N)$

$$\text{where } m_N = S_N(S_0^{-1}m_0 + \beta\Phi^T\mathbf{t}), S_N^{-1} = S_0^{-1} + \beta\Phi^T\Phi$$

- Maximum Likelihood:

- Likelihood : $p(\mathbf{t}|w) = \prod_{n=1}^N \mathcal{N}(t_n|\phi(x_n)w, \beta^{-1})$

- \ln Likelihood = $\sum_{n=1}^N [-\ln \frac{\beta}{\sqrt{2\pi}} - \frac{\beta}{2}(t_n - \phi(x)w)^2]$

- $\frac{\partial}{\partial w} \ln$ Likelihood = $\beta\Phi^T(\mathbf{t} - \Phi w)$

$$\text{let } \frac{\partial}{\partial w} \ln \text{ Likelihood} = 0$$

$$\Rightarrow w_{ML} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{t}$$

- $\frac{\partial}{\partial \beta} \ln$ Likelihood = $-N\beta^{\frac{1}{2}} + \beta^{\frac{3}{2}}(\mathbf{t} - \Phi w)^T(\mathbf{t} - \Phi w)$

$$\text{let } \frac{\partial}{\partial \beta} \ln \text{ Likelihood} = 0$$

$$\Rightarrow \beta^{-1} = \frac{1}{N}(\mathbf{t} - \Phi w)^T(\mathbf{t} - \Phi w)$$

Note: solve $w = w_{ML}$ first

- Maximum Posterior:
 - Posterior = $p(w|\mathbf{t})$, Prior = $p(w)$, Likelihood = $p(\mathbf{t}|w)$, Normalization = $p(\mathbf{t})$

$$\Rightarrow \text{Posterior} = \frac{\text{Likelihood} \cdot \text{Prior}}{\text{Normalization}}$$

$$\Rightarrow \text{Posterior} \propto \text{Likelihood} \cdot \text{Prior}$$
 - assume Prior $p(w) = \mathcal{N}(w|m_0, S_0)$,
so that Prior & Likelihood are conjugate \Rightarrow Gaussian Posterior
 - Likelihood $p(\mathbf{t}|w) = \prod_{n=1}^N \mathcal{N}(t_n|\phi(x_n)w, \beta^{-1}) = \mathcal{N}(\mathbf{t}|\Phi w, \beta^{-1}I)$
 - \Rightarrow Posterior $p(w|\mathbf{t}) = \mathcal{N}(w|m_N, S_N)$,
where $m_N = S_N(S_0^{-1}m_0 + \beta\Phi^T\mathbf{t})$, $S_N^{-1} = S_0^{-1} + \beta\Phi^T\Phi$
 $\Rightarrow w_{MAP}$ = mean of the Gaussian = m_N
Note: can also get w_{MAP} from taking gradient
- Simple Prior:

Prior $p(w) = \mathcal{N}(w|0, \alpha^{-1}I)$

\Rightarrow Posterior $p(w|\mathbf{t}) = \mathcal{N}(w|m_N, S_N)$,
where $m_N = \beta(\alpha I + \beta\Phi^T\Phi)^{-1}\Phi^T\mathbf{t}$, $S_N^{-1} = \alpha I + \beta\Phi^T\Phi$
 $w_{MAP} \rightarrow w_{ML}$, when $\alpha \rightarrow 0$ (most useless Prior)
- Maximize Posterior \Leftrightarrow Minimize cost function with regularization:

Simple Prior $\Rightarrow \ln p(w|\mathbf{t}) = -\frac{\beta}{2}(\mathbf{t} - \Phi w)^T(\mathbf{t} - \Phi w) - \frac{\alpha}{2}w^T w + \text{const}$
- If $\alpha \rightarrow 0$ (Prior is most useless), maximize Posterior is equivalent to maximizing likelihood
- Maximize Posterior equal to minimize sum-of-squares error function with the addition of a quadratic regularization term with $\lambda = \alpha/\beta$
- Regularization term comes from the Prior
- Predictive Distribution:
 - Assume: Prior : $p(x|\alpha) = \mathcal{N}(x|0, \alpha^{-1}I)$, ($m_0 = 0, S_0 = \alpha^{-1}I$)
 - $p(t|x, X, \mathbf{t}) = \int p(t|w, x)p(w|X, \mathbf{t})dw$
 - $\Rightarrow p(t|x, X, \mathbf{t}) = \mathcal{N}(t|m_N^T\phi(x), \sigma_N^2(x))$
where $\sigma_N^2(x) = \frac{1}{\beta} + \phi(x)^T S_N \phi(x)$; m_N, S_N from Posterior ($m_N = w_{MAP}$)
- Sequential data:
 - Posterior from previous data \Leftrightarrow the Prior for the arriving data

- Sequential data and data in one go is equivalent when finding the Posterior
- Gradient descent
 - Hypothesis function:
 - $h_\theta(x) = x\theta$, $\theta = [\theta_0, \theta_1, \dots, \theta_n]^T$, $x = [x_0, x_1, \dots, x_n]$, $x_0 = 1$
 - x is one instance
 - Cost function: $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^i) - y^i)^2 + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$
 - Update rule: $\forall \theta_j \in \theta$, $\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$, $\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m [(h_\theta(x^i) - y^i)x_j^i] + \frac{\lambda}{m} \theta_j$ - $\frac{d}{d\theta} J(\theta) = \frac{1}{m} ((X\theta - y)^T X)^T + \frac{\lambda}{m} [0, \theta_1, \dots, \theta_m]^T$ (θ_0 shouldn't be penalized)
 - simultaneously for all $\theta_j \in \theta$
 - Normal equation (mathematical solution)
 - $\theta = (X^T X)^{-1} X^T y$

2.4 Logistic Regression (Classification)

- Decision Theory:

- classes C_1, \dots, C_K , decision regions $\mathcal{R}_1, \dots, \mathcal{R}_K$ - Minimize $p(\text{mistake}) = \sum_{k=1}^K \left(\int_{\mathcal{R}_k} \sum_{i \neq k} p(x, C_i) dx \right)$ (can have weight on each misclassification though) - Maximize

$$p(\text{correct}) = \sum_{k=1}^K \int_{\mathcal{R}_k} p(x, C_k) dx$$

- Models for Decision Problems:

- Find a discriminant function - Discriminative Models: less powerful, yet less parameter = easier to learn

- Infer **posterior** $p(C_k|x)$, C_k : $x \in C_k$, x is examples in training set - Use decision theory to assign a new x - Generative Models: more powerful, but computationally expensive

- Infer conditional probabilities $p(x|C_k)$ - Infer prior $p(C_k)$ - Find **either** **posterior** $p(C_k|x)$, **or** **joint distribution** $p(x, C_k)$ (using Bayes' theorem) - Use decision theory to assign a new x - **= Able to create synthetic data using $p(x)$ **

- Naive Bayes on Discrete Features:

- Assumption:

- Discrete Features: data point $x \in \{0, 1\}^D$

- Naive Bayes: all features conditioned on class C_k are independent with each other

$$\Rightarrow p(x|C_k) = \prod_{i=1}^D \mu_{ki}^{x_i} (1 - \mu_{ki})^{1-x_i}$$

1. Linear Discriminant (Least Squares Approach)

- Prediction:

- $y(x) = xw + w_0$, with bias = w_0 , where $w = [w_1, \dots, w_n]^T$, $x = [x_1, \dots, x_n]$ - $y(x) > 0$: positive confidence to assign x to current class - $-w_0$ called threshold sometimes

- Decision Boundary $y(x) = w^T x + w_0 = 0$:

- w is orthogonal to vectors on the boundary:

assume x_1, x_2 on the boundary

$$\Rightarrow 0 = y(x_1) - y(x_2) = (x_1 - x_2)w$$

- Distance from origin to boundary is $-\frac{w_0}{\|w\|}$:

assume distance is k

$$\Rightarrow k \frac{w}{\|w\|} \text{ on boundary, thus } k \frac{w}{\|w\|} w + w_0 = 0$$

$$\Rightarrow k = -\frac{w_0}{\|w\|}$$

- $y(x)$ is a signed measure of distance from point x to boundary:

assume distance is r

$$\Rightarrow y(x) = \overbrace{(x_\perp + r \frac{w}{\|w\|})}^x w + w_0 = \overbrace{x_\perp w + w_0}^0 + r\|w\| = r\|w\|$$

$$\Rightarrow r = \frac{y(x)}{\|w\|}$$

- Multi-class (k-classes):

- prediction: x is of class C_k if $\forall j \neq k, y_k(x) > y_j(x)$

$\Rightarrow y(x) = xW$, where $W = [w_1, \dots, w_k]$, $\forall x_i \in X, x_{i0} = 1$ (bias), $y(x)$ is 1-of-k coding

- sum-of-squares error: $E_D(W) = \frac{1}{2} \text{tr}\{(XW - T)(XW - T)^T\}$

\Rightarrow optimal $W = (X^T X)^{-1} X^T T$

note that $\text{tr}\{AB\} = A^T B^T$

2. Fisher's Linear Discriminant

- Basic idea:

- Take linear classification $y = w^T x$ as dimensionality reduction (projection onto 1-D) - find a projection (denoted by vector w) which maximally preserves the class separation - if $y > -w_0$ then class C_1 , otherwise C_2

- Goal:

- Distance within one class is small - Distance between classes is large

- Mean & Variance of Projected Data:

- Mean: $\tilde{m}_k = w^T m_k$, where $m_k = \frac{1}{N_k} \sum_{x \in C_k} x$ - Variance: $\tilde{s}_k = \sum_{x \in C_k} (w^T x -$

$$w^T m_k)^2 = w^T \left[\sum_{x \in C_k} (x - m_k)(x - m_k)^T \right] w$$

- 2-Class to 1-D line:

- Maximize Fisher criterion: $J(w) = \frac{|\tilde{m}_1 - \tilde{m}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$

- Between-class covariance: $S_B = (m_1 - m_2)(m_1 - m_2)^T$

- Within-class covariance: $S_k = \sum_{n \in C_k} (x_n - m_k)(x_n - m_k)^T$

$$\Rightarrow J(w) = \frac{w^T S_B w}{w^T S_W w}$$

- Lagrangian: $L(w, \lambda) = w^T S_B w + \lambda(1 - w^T S_W w)$

fix $w^T S_W w$ to 1 to avoid infinite solution (any multiple of a solution is a solution)

$$\Rightarrow \frac{\partial}{\partial w} L = 2S_B w - 2\lambda S_W w = 0$$

$$\Rightarrow S_B w = \lambda S_W w$$

$$\Rightarrow (S_W^{-1} S_B) w = \lambda w$$

To maximize $J(w)$, w is the largest eigenvector of $S_W^{-1} S_B$ if S_W invertible

- K-classes to a d-D subspace: N_k is num in class k, N is the total example num

- Between-class covariance: $S_B = \sum_{k=1}^K N_k (m_k - m)(m_k - m)^T$, where $m =$

$$\frac{1}{N} \sum_{n=1}^N x_n$$

reduce to $(m_1 - m_2)(m_1 - m_2)^T$ when $K=2$ (constant ignored)

- Within-class covariance: $S_W = \sum_{k=1}^K S_k$, where $S_k = \sum_{n \in C_k} (x_n - m_k)(x_n -$

$$m_k)^T, m_k = \frac{1}{N_k} \sum_{n \in C_k} x_n$$

- Maximize Fisher criterion: $J(w) = \frac{\text{tr}\{W^T S_B W\}}{\text{tr}\{W^T S_W W\}}$

- Lagrangian:

Solve for each $w_i \in W \Rightarrow (S_W^{-1} S_B) w_i = \lambda_i w_i$

$\Rightarrow W$ consists of the largest d eigenvectors

$S_W^{-1} S_B$ is not guaranteed to be symmetric $\Rightarrow W$ might not be orthogonal

Need to minimize the whole covariance matrix ($J(w)$ as a matrix) \Rightarrow not choosing same eigenvectors twice

- Maximum Possible Projection Directions = $K - 1$:

$$r(S_W^{-1} S_B) \leq \min(r(S_W^{-1}), r(S_B)) \leq r(S_B)$$

$$r(S_B) \leq \sum_K r((m_k - m)(m_k - m)^T) = K, \text{ as } r(m_k - m) = 1$$

$$\sum_K m_k = m \Rightarrow r(m_1 - m, \dots, m_K - m) = K - 1$$

$$\Rightarrow r(S_B) \leq K - 1$$

$$\Rightarrow r(S_W^{-1} S_B) \leq K - 1$$

3. Perceptron Algorithm

- Generalised linear model $y = f(w^T \phi(x))$, where $\phi(x)$ is basis function;

$$\phi_0(x) = 1$$

- Nonlinear activation function: $f(a) = \begin{cases} 1, & a \geq 0 \\ -1, & a < 0 \end{cases}$

- Target coding: $t = \begin{cases} 1, & \text{if } C_1 \\ -1, & \text{if } C_2 \end{cases}$

- Cost function:

- All correctly classified patterns: $w^T \phi(x_n) t_n > 0$
- Add the errors for all misclassified patterns (denoted as set \mathcal{M}):
 $\Rightarrow E_P(w) = - \sum_{n \in \mathcal{M}} w^T \phi(x_n) t_n$
- Algorithm: (Aim: minimize total num of misclassified patterns)
- loop
- choose a training pair (x_n, t_n)
- update the weight vector w : $w = w - \eta \nabla E_P(w) = w + \phi_n t_n$
- where $\eta=1$ because $y = f(\cdot)$ does not depend on $\|w\|$
- Perceptron Convergence Theorem:
- If the training set is linearly separable, the perceptron algorithm is guaranteed to find a solution in a finite number of steps
- (Also is the algorithm to find whether the set is linearly separable = Halting Problem)

4. Maximum Likelihood

- Assumption: - $p(x|C_k) \sim \mathcal{N}(\mu_k, \Sigma)$, and all $p(x|C_k)$ share the same Σ -
 - $p(C_1) = \pi, p(C_2) = 1 - \pi$, π unknown - Likelihood of whole data set \mathbf{X}, \mathbf{t} , N is
- the num of data - $p(\mathbf{X}, \mathbf{t} | \pi, \mu_1, \mu_2, \Sigma) = \prod_{n=1}^N [\pi \mathcal{N}(x_n | \mu_1, \Sigma)]^{t_n} [(1-\pi) \mathcal{N}(x_n | \mu_2, \Sigma)]^{1-t_n}$

\rightarrow when info of label t lost: mixture of Gaussian - $\ln(\text{Likelihood}) = \sum_{n=1}^N [t_n (\ln \pi + \ln \mathcal{N}(x_n | \mu_1, \Sigma)) + (1-t_n) (\ln(1-\pi) + \ln \mathcal{N}(x_n | \mu_2, \Sigma))]$ - Parameters when maximum reached: - $\pi = \frac{N_1}{N_1 + N_2}$, N_1 is the num of class C_1 - $\mu_1 = \frac{1}{N_1} \sum_{n=1}^N t_n x_n$, $\mu_2 =$

$$\frac{1}{N_2} \sum_{n=1}^N (1-t_n) x_n, \text{ (mean of each class) - } \Sigma = \frac{N_1}{N} S_1 + \frac{N_2}{N} S_2, \text{ where } S_k = \frac{1}{N_k} \sum_{n \in C_k} (x_n - \mu_k)(x_n - \mu_k)^T$$

5. Logistic Regression

- Sigmoid function: $\sigma(a) = \frac{1}{1 + e^{-a}}$
- $p(x|C_k) \sim \mathcal{N} \Rightarrow p(C_k|x) = \sigma(w^T x + w_0)$ (2-classes) (Generative model)
- Assumption:
- $p(x|C_k) = \mathcal{N}(\mu_k, \Sigma)$ (can also be a number of other distributions)
- $\forall k, p(x|C_k)$ shares the same Σ

$$p(C_1|x) = \frac{p(x|C_1)p(C_1)}{p(x|C_1)p(C_1) + p(x|C_2)p(C_2)} = \sigma(a),$$

$$\begin{aligned} \text{where } a &= \ln \frac{p(x, C_1)}{p(x, C_2)} \\ &= \ln \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)} \\ &= \dots (\text{assumption applied}) \\ &= \ln \frac{\exp(\mu_1^T \Sigma^{-1} x - \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1)}{\exp(\mu_2^T \Sigma^{-1} x - \frac{1}{2} \mu_2^T \Sigma^{-1} \mu_2)} + \ln \frac{p(C_1)}{p(C_2)} \\ \implies a &= w^T x + w_0 \text{ where,} \\ w &= \Sigma^{-1}(\mu_1 - \mu_2) \\ w_0 &= -\frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1 + \frac{1}{2} \mu_2^T \Sigma^{-1} \mu_2 + \ln \frac{p(C_1)}{p(C_2)} \end{aligned}$$

$$- \implies p(C_1|x) = \sigma(w^T x + w_0)$$

\Rightarrow Find parameters in Gaussian model using Maximal Likelihood Solution

as: $p(C_1|x) \propto p(x|C_1)p(C_1) = p(x, C_1)$

- Generalize to K-classes:

$$a_k(x) = \ln[p(x|C_k)p(C_k)], p(C_k|x) = \frac{\exp(a_k)}{\sum_i \exp(a_i)}$$

$$\Rightarrow a_k(x) = w_k^T x + w_{k0}, \text{ where } w_k = \Sigma^{-1} \mu_k; w_{k0} = -\frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + p(C_k)$$

- Assume directly $p(C_k|x) = \sigma(w^T x + w_0)$ (2-classes) (Discriminative model)

- Assume directly: $p(C_1|w, x) = \sigma(w^T x), x_0 = 1$

\Rightarrow less parameters to fit (compared to Gaussian)

- Likelihood function:

$$p(\mathbf{t}|w, X) = \prod_{n=1}^N p_n^{t_n} (1 - p_n)^{1-t_n}, \text{ where, } p_n = p(C_1|x_n), t_n \text{ is the class of } x_n$$

Define error function :

$$E(w) = -\ln(\text{Likelihood}) = -\sum_{n=1}^N [t_n \ln p_n + (1 - t_n) \ln(1 - p_n)]$$

$$\Rightarrow \nabla E(w) = \sum_{n=1}^N (p_n - t_n) x_n$$

- Find Posterior $p(w|\mathbf{t})$:

Likelihood is product of sigmoid

Conjugate Prior for "sigmoid distribution" is unknown

\Rightarrow Assume Prior $p(w) = \mathcal{N}(w|m_0, S_0)$

$$\Rightarrow \ln p(w|\mathbf{t}) \propto -\frac{1}{2} (w - m_0)^T S_0^{-1} (w - m_0) + \sum_{n=1}^N [t_n \ln p_n + (1 - t_n) \ln(1 - p_n)]$$

find w_{MAP} , calculate $S_N = -\nabla\nabla \ln p(w|\mathbf{t}) = S_o^{-1} + \sum_{n=1}^N p_n(1-p_n)\phi_n\phi_n^T$
 $\Rightarrow p(w|\mathbf{t}) \simeq \mathcal{N}(w|w_{MAP}, S_N)$, via Laplace Approximation
 - Laplace Approximation:
 - Fit a gaussian to $p(z)$ at its ****mode**** (mode of $p(z)$: point where $p'(z) = 0$)
 - Assume $p(z) = \frac{1}{Z}f(z)$, with normalization $Z = \int f(z)dz$
 Taylor expansion of $\ln f(z)$ at z_0 : $\ln f(z) \simeq \ln f(z_0) - \frac{1}{2}A(z-z_0)^2$,
 where $f'(z_0) = 0$, $A = -\frac{d^2}{dz^2} \ln f(z)|_{z=z_0}$
 Take its exponentiating: $f(z) \simeq f(z_0)\exp-\frac{A}{2}(z-z_0)^2$
 \Rightarrow Laplace Approximation = $(\frac{A}{2\pi})^{1/2}\exp-\frac{A}{2}(z-z_0)^2$, where $A = \frac{1}{\sigma^2}$
 - Requirement:
 $f(z)$ differentiable to find a critical point
 $f''(z_0) < 0$ to have a maximum & so that $\nabla\nabla \ln f(z_0) = A > 0$ as $A = \frac{1}{\sigma^2}$
 - In Vector Space: approximate $p(z)$ for $z \in \mathcal{R}^M$
 Assume $p(z) = \frac{1}{Z}f(z)$
 Taylor expansion: $\ln f(z) \simeq \ln f(z_0) - \frac{1}{2}(z-z_0)^T A(z-z_0)$,
 Hessian $A = -\nabla\nabla \ln f(z)|_{z=z_0}$

$$\Rightarrow \text{Laplace approximation} = \frac{|A|^{1/2}}{(2\pi^{M/2})} \exp - \frac{1}{2}(z-z_0)^T A(z-z_0) \quad (2.1)$$

$$= \mathcal{N}(z|z_0, A^{-1}) \quad (2.2)$$

- Gradient descent:
 - Hypothesis function: $h_\theta(x) = \sigma(x\theta) = \frac{1}{1+e^{-x\theta}}$
 - Cost function:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m [-y^i \ln(h_\theta(x^i)) - (1-y^i) \ln(1-h_\theta(x^i))] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

 - Update rule: $\forall \theta_j \in \theta, \theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}, \frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m [(h_\theta(x^i) - y^i)x_j^i] + \frac{\lambda}{m} \theta_j$

2.5 Diagnose Machine Learning

- Size of data set - value of λ - Num of feature - Degree of polynomial - Build a quick implmentation

2.5.1 Evaluating Hypothesis

1. Data set =, training set (60)
 - randomly split

2. Cross Validation:

- estimation of the generalization error, inaccurate because:

- finite training set - finite cross validation set

- S-fold Cross Validation:

- divide whole set of data into S sets

- for $i \in (1, S)$

choose the i^{th} set as cross validation (or test set in some cases)

rest of the sets are called training set

run the procedure (mentioned later) on the training set

estimate generalization in CV set

3. Test set error

- Linear regression: $J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\theta}(x_{test}^i) - y_{test}^i)^2$ - Logistic re-

gression: $J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} (y_{test}^i \log h_{\theta}(x_{test}^i) + (1 - y_{test}^i) \log (1 - h_{\theta}(x_{test}^i)))$ -

Misclassification error: $J_{test}(\theta) = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_{\theta}(x_{test}^i), y_{test}^i)$ -

$$err(h_{\theta}(x), y) = \begin{cases} 1 & \text{if } h_{\theta}(x) \geq 0.5, y = 0 \text{ or } h_{\theta}(x) < 0.5, y = 1 \\ 0 & \text{otherwise} \end{cases} \quad (2.3)$$

4. Choosing procedure:

- Minimize training error $J_{train}(\theta)$ - Select a model with lowest $J_{cv}(\theta)$ -

Estimate generalization error as $J_{test}(\theta)$

2.5.2 Bias / Variance

1. Problems:

- High bias: underfitting - high $J_{train}(\theta)$ and high $J_{cv}(\theta)$ - High variance: overfitting - low $J_{train}(\theta)$ but high $J_{cv}(\theta)$

2. Interaction with regularization:

- Improper λ : - large λ = high bias - small λ = high variance - Choosing λ : - try $\lambda = 0, 0.01, 0.02, 0.04, \dots, 10$ - select the model with lowest $J_{cv}(\theta)$ without regularization term

3. Interaction with training set size:

- Normal Learning curve:

! [Normal learning curve] (../Machine

- Learning curve with high bias:

- where getting more training data ****doesn't**** help

! [Learning curve with high bias] (../Machine

- Learning curve with high variance:

- where getting more training data ****helps****

! [Learning curve with high variance] (../Machine

4. Ways to fix:

- High bias: - more features / more polynomial terms of features - decreasing λ
- High variance:
 - larger data set - fewer features - increasing λ
 - **In neural network:**
 - High bias \Rightarrow larger neural networks (more hidden layers / more units in one layer)
 - High variance \Rightarrow smaller neural networks
 - **Larger network with regularization (λ) is more powerful**

2.5.3 Error Analysis

1. Procedure: - Algorithm (trained) misclassifies n data in cross validation set - Classify these n data and rank them - Maybe more features are found
2. Feature selection \Rightarrow Numerical evaluation - \Rightarrow test algorithm with / without this feature on **CV set** (compare error rate)

2.5.4 Skewed classes

1. Precision / Recall

- **Actual** 1 **Actual** 0

- **Predicted** 1 True positive False positive **Predicted** 0 False negative True negative

- **Precision** = $\frac{\text{True positive}}{\text{Predicted positive}} = \frac{\text{True positive}}{\text{True pos} + \text{False pos}}$

- **Recall** = $\frac{\text{True positive}}{\text{Actual positive}} = \frac{\text{True positive}}{\text{True pos} + \text{False neg}}$
2. Evaluation with precision/recall
 - Predict 1 if $h_\theta(x) \geq \epsilon$, 0 if $h_\theta(x) < \epsilon$
 - larger $\epsilon \Rightarrow$ higher precision, lower recall (more confident) - smaller $\epsilon \Rightarrow$ lower precision, higher recall (avoid missing)
 - [\[Possible Precision-Recall curve\]\(../Machine Learning/Statistical Machine Learning/Possible Precision-Recall curve.png\)](#)
3. Compare precision/recall num
 - $F_1 \text{ Score} = 2 \frac{PR}{P+R}$, P as precision, R as recall - higher better, on cross validation set
4. High precision & high recall:
 - **large num of features (low bias) + large sets of data (low variance)**

2.6 Latent Variable Analysis

2.6.1 Principal Component Analysis (PCA)

1. Motivation:
 - Data compression (reduce highly related features) - Data visualization

- 2. Assumption:
 - Gaussian distributions for both the latent and observed variables
- 3. Two Equivalent Definition of PCA:
 - Linear projection of data onto lower dimensional linear space (principal subspace) such that:
 - \Rightarrow variance of projected data is maximized
 - \Rightarrow distortion error from projection is minimized
- 4. Maximum Variance Formulation
 - Goal:
 - project data from D dimension to M while maximizing the variance of projected data
 - Eigenvalues λ of covariance matrix S express the variance of data set X in direction of corresponding eigenvectors
 - Projection Vectors:
 - $U = (u_1, \dots, u_M)$, where $\forall i \in \{1, \dots, M\}, u_i \in \mathbb{R}^D$ s.t. $u_i^T u_i = 1$ (only consider direction)
 - Projected Data:
 - Mean = $\bar{x}^T U$, where $\bar{x} = \frac{1}{N} \sum_{i=1}^N x^i$ - Variance = $\text{tr}\{U^T S U\}$, where $S = \sum_{i=1}^N (x^i - \bar{x})(x^i - \bar{x})^T$ (outer product)
 - Lagrangian to maximize Variance:
 - $L(U, \lambda) = \text{tr}\{U^T S U\} + \text{tr}\{(I - U^T U)\lambda\}$
 - constraint $u_i^T u_i = 1$ to prevent $u_i \rightarrow +\infty$

$$\text{For each } u_i \in U, \frac{\partial}{\partial u_i} L = 2S u_i - 2\lambda_i u_i = 0 \quad (2.4)$$

$$\Rightarrow S u_i = \lambda_i u_i \quad (2.5)$$

$$\Rightarrow U \text{ consists of eigenvectors corresponding to the first } M \text{ large eigenvalue of } S \quad (2.6)$$

(S symmetric $\Rightarrow U$ orthogonal)

5. Minimum Error Formulation:

- Introduce Orthogonal Basis Vector for D dimension:
 - $U = (u_1, \dots, u_D)$
- Data representation:
 - Original: $x^n = \sum_{i=1}^D \alpha_i^n u_i$ - Projected: $\widetilde{x}^n = \sum_{i=1}^M z_i^n u_i + \sum_{i=M+1}^D b_i u_i$
 - (z_1^n, \dots, z_M^n) is different for different x^n , (b_{M+1}, \dots, b_D) is the same for all x^n
- Cost function: $J = \frac{1}{N} \sum_{n=1}^N \|x^n - \widetilde{x}^n\|^2$, where $\widetilde{x}^n = \sum_{i=1}^M z_i^n u_i + \sum_{i=M+1}^D b_i u_i$

$$\begin{aligned}
\text{- Let } \begin{cases} \frac{\partial}{\partial z_j^n} J = 0 \\ \frac{\partial}{\partial b_j} J = 0 \end{cases} &\Rightarrow \begin{cases} \frac{1}{N} 2(x^n - \widetilde{x}^n)^T (-u_j) = \frac{2}{N} (z_j - (x^n)^T u_j) = 0 \\ \frac{1}{N} \sum_{n=1}^N 2(x^n - \widetilde{x}^n)^T (-u_j) = \frac{2}{N} \sum_{n=1}^N (b_j - (x^n)^T u_j) = 0 \end{cases} \\
\Rightarrow \begin{cases} z_j = (x^n)^T u_j & j \in \{1, \dots, M\} \\ b_j = \bar{x}^T u_j & j \in \{M+1, \dots, D\} \end{cases}
\end{aligned}$$

$$\text{Noticing } (x^n)^T u_j = \left(\sum_{i=1}^D \alpha_i^n u_i^T \right) u_j = a_j \Rightarrow a_j = (x^n)^T u_j$$

$$\Rightarrow x^n - \widetilde{x}^n = \sum_{i=M+1}^D [(x^n - \bar{x})^T u_i] u_i$$

$$\Rightarrow J = \frac{1}{N} \sum_{n=1}^N \left(\sum_{i=M+1}^D [(x^n - \bar{x})^T u_i] u_i \right)^T \left(\sum_{i=M+1}^D [(x^n - \bar{x})^T u_i] u_i \right) \quad (2.7)$$

$$= \frac{1}{N} \sum_{n=1}^N \left(\sum_{i=M+1}^D u_i^T ((x^n - \bar{x})^T u_i) \right) \left(\sum_{i=M+1}^D ((x^n - \bar{x})^T u_i) u_i \right) \quad (2.8)$$

$$= \frac{1}{N} \sum_{n=1}^N \sum_{i=M+1}^D u_i^T (x^n - \bar{x})^T u_i u_i^T (x^n - \bar{x}) u_i \quad u_i \text{ orthogonal to each other} \quad (2.9)$$

$$= \sum_{i=M+1}^D u_i^T \left(\frac{1}{N} \sum_{n=1}^N (x^n - \bar{x})^T (x^n - \bar{x}) \right) u_i \quad \|u_i\| = 1 \quad (2.10)$$

$$(2.11)$$

$$\Rightarrow J = \sum_{i=M+1}^D u_i^T S u_i, \text{ where } S = \frac{1}{N} \sum_{n=1}^N (x^n - \bar{x})^T (x^n - \bar{x})$$

- Lagrangian to Minimize J :

$$- L(u_{M+1}, \dots, u_D, \lambda_{M+1}, \dots, \lambda_D) = \sum_{i=M+1}^D u_i^T S u_i + \sum_{i=M+1}^D \lambda_i (1 - u_i^T u_i)$$

constraint $\|u_i\| = 1$ to prevent $u_i = 0$

$$\text{For each } u_i, \frac{\partial}{\partial u_i} L = 2S u_i - 2\lambda_i u_i = 0$$

$$\Rightarrow S u_i = \lambda_i u_i$$

\Rightarrow To minimize J , take eigenvectors with the first $(D - M)$ small eigenvalue orthogonal to (out of) subspace

\Leftrightarrow define subspace with eigenvectors with the first M large eigenvalue

$$\text{Intuition: } \widetilde{x}_n = \sum_{i=1}^M ((x^n)^T u_i) u_i + \sum_{i=M+1}^D (\bar{x}^T u_i) u_i \quad (2.12)$$

$$= \bar{x} + \sum_{i=1}^M [(x^n - \bar{x})^T u_i] u_i \quad (2.13)$$

1. Singular Value Decomposition - SVD:

- Introduce matrix $A_{m \times n}$
- $(A^T A)_{n \times n}$ symmetric matrix (actually, Gram matrix \rightarrow semi-definite) -

eigenvalue decomposition: (2.14)

$A^T A = V D V^T$, V is normalized ($v_i^T v_i = 1$) with column as eigenvector (2.15)

- $AV = (Av_1, \dots, Av_n)_{m \times n}$
- Let $r(A) = r$

$$\Rightarrow r(A^T A) = r(A) = r \quad (2.16)$$

$$r(AV) = \min\{r(A), r(V)\} = \min\{r, n\} = r \quad (2.17)$$

- Reduce AV to basis (Av_1, \dots, Av_r)
- Let $U = (u_1, \dots, u_r) = (\frac{Av_1}{\sqrt{\lambda_1}}, \dots, \frac{Av_r}{\sqrt{\lambda_r}})$, λ_i is i -thh eigenvalue of $A^T A$
- Orthogonal: $\forall i \neq j, u_i^T u_j = \frac{1}{\sqrt{\lambda_i \lambda_j}} v_i^T A^T A v_j = \frac{\lambda_j}{\sqrt{\lambda_i \lambda_j}} v_i^T v_j = 0$
- Unit: $\|u_i\| = \frac{\|Av_i\|}{\sqrt{\lambda_i}} = \frac{\sqrt{\langle Av_i, Av_i \rangle}}{\sqrt{\lambda_i}} = 1$
- $\Rightarrow U$ is standard orthogonal (orthonormal) basis
- $AV = U \Sigma$, where $\Sigma = D^{\frac{1}{2}}$
- Expand U to orthonormal in $\mathbb{R}^m : (u_i, \dots, u_m)$
- Expand corresponding part in Σ with 0
- $A = U \Sigma V^T$, with singular value in Σ in decreasing order

2. SVD with PCA:

- X is data matrix in row (centered - zero mean)
- Eigenvectors of covariance matrix $S = X^T X$ are in V , where $X = U \Sigma V^T$
- When using $S = U \Sigma V^T \Rightarrow U = V \wedge S = V \Sigma V^T$

reduced to eigenvalue decomposition

- $S = V D V^T$ with V orthonormal:

Eigenvalues λ of covariance matrix S express the variance of data set X in direction of corresponding eigenvectors

- Projection:

- $\tilde{X} = X V_M$, where V_M contains first M -large eigenvectors - Projection direction is **not** unique

3. Reconstruction (approximate):

- Data is projected onto k dimension using SVD with $S = U\Sigma V^T$ - $x_{approx} = U_{reduce} \cdot z$, U_{reduce} is $n \times k$ matrix, z is $k \times 1$ vector - [Reconstruction from data Compression](../Machine

4. Choosing k (num of principal components):

- choose the **smallest** k making $\frac{J}{V} \leq 0.01 \Rightarrow 99$

- $[U, S, V] = \text{svd}(\text{Sigma}) \Rightarrow \frac{J}{V} = 1 - \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}}$, S is diagonal matrix

\Rightarrow check $\frac{J}{V}$ before compress data

5. Data Preprocessing:

- PCA *vs.* Normalization: - Normalization: Individually normalized but still correlated - PCA: create decorrelated data - whitening - Whitening: projection with normalization - $S = VDV^T$, where S is Gram matrix over X^T - $\forall n, y_n = D^{-\frac{1}{2}} V^T (x^n - \bar{x})$, where \bar{x} is the mean of X

$\Rightarrow y^n$ has zero mean (2.18)

$$\text{cov}(\{y^n\}) = \frac{1}{N} \sum_{n=1}^N y_n y_n^T = D^{-\frac{1}{2}} V^T S V D^{-\frac{1}{2}} = I \quad (2.19)$$

6. Tips for PCA:

- Do NOT use PCA to prevent overfitting, use regularization instead - Try original data before implement PCA - Train PCA only on training set

2.6.2 Independent Component Analysis (ICA)

1. Goal: - Recover original signals from a mixed observed data - Source signal $S \in \mathbb{R}^{N \times K}$; mixing matrix A ; Observed data $X = SA$ - Maximizes statistical independence - Find A^{-1} to maximizes independence of columns of S 2. Assumption: - At most one signal is Gaussian distributed - Ignore amplitude and order of recovered signals - Have at least as many observed mixtures as signals - A invertible 3. Independence *vs.* Uncorrelatedness - Independence \Rightarrow Uncorrelatedness - $p(x_1, x_2) = p(x_1)p(x_2) \Rightarrow \mathbb{E}(x_1 x_2) - \mathbb{E}(x_1)\mathbb{E}(x_2) = 0$ 4. Central Limit Theorem 5. FastICA algorithm

2.6.3 t-SNE

1. Problem & Focus 2. Compared to PCA: - No whitening function to use for new data - PCA can only capture linear structure inside the data - t-SNE preserves the [local distances] in the original data

2.6.4 Anomaly Detection

1. Problem to solve:

- Given dataset x^1, x^2, \dots, x^m , build density estimation model $p(x)$ - $p(x^{test} < \epsilon) \Rightarrow x^{test}$ anomaly

2. Hypothesis function:

$$- p(x) = \prod_{i=1}^n p(x_i), x \in R^n, \forall i \in [1, n], x_i \sim N(\mu_i, \sigma_i^2) - \mu = \frac{1}{m} \sum_{i=1}^m x^i, \sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^i - \mu)^2 - \text{assume } x_1, \dots, x_n \text{ independent from each other}$$

3. Multivariate Gaussian:

$$- p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right),$$

$$x \in R^n, \mu \in R^n, \Sigma \in R^{n \times n}, \text{ where } \Sigma \text{ is covariance matrix}$$

$$- \mu = \frac{1}{m} \sum_{i=1}^m x^i, \Sigma = \frac{1}{m} \sum_{i=1}^m (x^i - \mu)(x^i - \mu)^T - x_1, \dots, x_n \text{ can be correlated}$$

but **not** linearly dependent - need $m > n$ ($m \geq 10n$ suggested) or else Σ non-invertible

4. Algorithm:

- choose features - compute μ, σ - compute $p(x)$ for new example, anomaly if $p(x) < \epsilon$

5. Evaluation (real-number):

- Labeled data into normal/anomalous set
(okay if some anomalies slip into normal set)
- training set: unlabeled data from normal set (60- CV set: labeled data from normal (20- test set: labeled data from normal (20
- Use evaluation metrics (skewed data)

6. When to use:

- Anomaly detection: - Very small num of positive data (0-20 commonly); Large num of negative data - Difficult to learn from positive data (not enough data, too many features...) - Future anomalies may look nothing like given data - Supervised Learning: - Larger num of positive & negative data - Enough positive data for algorithm to learn - Future positive example is likely to be similar to given data

7. Example:

- Anomaly detection: - Fraud detection, Manufacturing, Monitoring machines in data center... - Supervised learning: - Email spam classification (enough data), Weather prediction (sunny/rainy/etc), Cancer classification...

8. Tips:

- Non-gaussian feature: transformation / using other distribution - Choosing features: compare anomaly data with normal data

2.6.5 Recommender System

1. Problem Formulation:

- $r_{i,j} = 1$ if item i is rated by user j
- $y_{i,j}$ = rating of item i given by user j
- θ^j = parameter vector for user j
- x^i = feature vector for movie i
- = $\hat{y}_{i,j}$ for user j , movie i , ($r_{i,j} = 0$), predict rating $x^i \theta^j$

2. Content Based Recommendations:

- Treat each user as a separate linear regression problem with the feature vectors of its rated items as training set

Assume features for each items (x^i) are available and known

=, given X estimate Θ

- Cost Function for θ_j :

$$J(\theta^j) = \frac{1}{2} \sum_{i:r_{i,j}=1} (x^i \theta^j - y_{i,j})^2 + \frac{\lambda}{2} \sum_{k=1}^n (\theta_k^j)^2, \theta^j \in R^{n+1} (\theta_0 \text{ not regularized})$$

- Cost Function for Θ :

$$J(\Theta) = \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r_{i,j}=1} (x^i \theta^j - y_{i,j})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^j)^2,$$

$\theta^j \in R^{n+1}$ (θ_0 not regularized), n_u is num of users

- Update Rule: $\forall \theta_k^j \in \theta^j, \theta_k^j := \theta_k^j - \alpha \frac{\partial J(\Theta)}{\partial \theta_k^j}, \frac{\partial J(\Theta)}{\partial \theta_k^j} = \sum_{i:r_{i,j}=1} (x^i \theta^j - y_{i,j}) x_k^i +$

$\lambda \theta_k^j$, for $k \neq 0$ ($\theta^j \in R^{n+1}$)

3. Collaborative Filtering

- **Assume preference of each users (θ^j) are available and known**

=, given Θ estimate X

- Cost Function for x^i : $J(x^i) = \frac{1}{2} \sum_{j:r_{i,j}=1} (x^i \theta^j - y_{i,j})^2 + \frac{\lambda}{2} \sum_{k=1}^n (x_k^i)^2$ - Cost

$$\text{Function for } X: J(X) = \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r_{i,j}=1} (x^i \theta^j - y_{i,j})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^i)^2$$

$x^j \in R^{n+1}$ (x_0 not regularized), n_m is num of items - Update Rule: $\forall x_k^i \in x^i, x_k^i :=$

$$x_k^i - \alpha \frac{\partial J(X)}{\partial x_k^i}, \frac{\partial J(X)}{\partial x_k^i} = \sum_{j:r_{i,j}=1} (\theta^j x^i - y_{i,j}) \theta_k^j + \lambda x_k^i, \text{ for } k \neq 0 (x^i \in R^{n+1})$$

- Basic Idea:

- Randomly initialize Θ

- loop:

Estimate X

Estimate Θ

- Cost Function:

$$J(X, \Theta) = \frac{1}{2} \sum_{(i,j):r_{i,j}=1} (x^i \theta^j - y_{i,j})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^i)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^j)^2, x \in$$

$R^n, \theta \in R^n$

(the sum term in $J(\Theta)$, $J(X)$, and $J(X, \Theta)$ is the same)

- Update Rule:

$$\forall x_k^i \in x^i, x_k^i := x_k^i - \alpha \frac{\partial J(X, \Theta)}{\partial x_k^i}, \frac{\partial J(X, \Theta)}{\partial x_k^i} = \frac{\partial J(X)}{\partial x_k^i} = \sum_{j:r_{i,j}=1} (\theta^j x^i -$$

$$y_{i,j}) \theta_k^j + \lambda x_k^i, x^i \in R^n - \forall \theta_k^j \in \theta^j, \theta_k^j := \theta_k^j - \alpha \frac{\partial J(X, \Theta)}{\partial \theta_k^j}, \frac{\partial J(X, \Theta)}{\partial \theta_k^j} = \frac{\partial J(\Theta)}{\partial \theta_k^j} =$$

$$\sum_{i:r_{i,j}=1} (\theta^j x^i - y_{i,j}) x_k^i + \lambda \theta_k^j, \theta^j \in R^n$$

- **Algorithm**
- Initialize X, Θ to **small random values**
- => for symmetry breaking (similar to random initialization in neural network)
- => so that algorithm learns features x^1, \dots, x^{n_m} that are different from each other
- Minimize $J(X, \Theta)$
- Predict $y_{i,j} = x^i \theta^j$ ($Y = X\Theta$)
- Finding Related Item to Recommend
- $\|x^i - x^j\|$ is small => item i and j is similar
- Mean Normalization:
- Problem: if user j hasn't rated any movie, $\theta^j = [0, \dots, 0]$
- => predicted rating of user j on all item = 0
- => useless prediction
- Algorithm (row version):
- compute vector $\mu, \forall \mu_i \in \mu, \mu_i = \text{mean of } Y_i$, where Y_i is the i^{th} row in Y
- manipulate Y : $\forall y_{i,j} \in Y \wedge r_{i,j} = 1, y_{i,j} - \mu_i = \tilde{y}_{i,j}$ the mean of each row in Y is 0
- predict rating for user j on item $i = x^i \theta^j + \mu_i$
- For item i with no rating
- => apply column version of mean normalization
- (but user with no rating is generally more important)

2.7 Large Scale Machine Learning

2.7.1 Gradient Descent with Large Dataset

1. Stochastic Gradient Descent - Problem in Big Data: - Updating θ becomes computationally expensive in batch gradient decent

- Cost Function: - Cost function on single data: $cost(\theta, (x^i, y^i)) = \frac{1}{2} (h_\theta(x^i) - y^i)^2$

- Overall Cost Function: $J_{train}(\theta) = \frac{1}{m} \sum_{i=1}^m cost(\theta, (x^i, y^i))$

- Procedure:
- Randomly shuffle dataset
- Repeat
- for $i \in [1, m]$

$$\begin{aligned}\theta_j &= \theta_j - \alpha \frac{\partial}{\partial \theta_j} \text{cost}(\theta, (x^i, y^i)) \\ &= \theta_j - \alpha (h_\theta(x^i) - y^i) \cdot x_j^i\end{aligned}\quad (2.20)$$

$$(\text{for } j = 0, \dots, n) \quad (2.21)$$

$$\Rightarrow \text{make progress with each single data} \quad (2.22)$$

- Convergence:
- Wanting θ to converge \Rightarrow slowly decrease α over time (but more parameters)

$$(\text{E.g } \alpha = \frac{\text{const}_1}{\text{iteration num} + \text{const}_2})$$

- Compute $\text{cost}(\theta, (x^i, y^i))$ before updating
- For every k update iterations, plot average $\text{cost}(\theta, (x^i, y^i))$ over the last k examples

- Checking curves:
- Increasing k result in smoother line and less noise, but the result is more delayed

Use smaller learning rate α will generally have slight benefit

Curve goes up \Rightarrow smaller α

- *vs* Batch Gradient Descent:
- use 1 example un each update iteration \Rightarrow make progress earlier \Rightarrow faster
- Result may not be the optimal but in its neighbourhood

1. Mini-batch Gradient Descent - Use b examples in each update iteration - *vs* Batch Gradient Descent: - start to make progress earlier \Rightarrow faster - Result may not be the optimal but in its neighbourhood - *vs* Stochastic Gradient Descent: - can partially parallelize computation over b examples \Rightarrow faster under a good vectorized implementation & appropriate b - introduce extra parameter b

2.7.2 Online Learning

1. Situation: - Has too many data (can be considered as infinite) - When data comes in as a continuous stream - Can adapt to changing user preference 2. Procedure: - Use one example only once (Similar to stochastic gradient decent in this sense)

2.7.3 Map-reduce

1. In Batch Gradient Descent:

- Update rule $\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^i) - y^i) x_j^i$ - Parallelize the computation

of $\sum_{i=1}^m (h_\theta(x^i) - y^i) x_j^i$ by dividing the data set into multiple sections

2. Ability to reduce:
 - Contain operation over the whole data set
 - (Neural Network can be map-reduced)

2.8 Building Machine Learning System

- Under the example of Photo OCR (Optical Character Recognition)

2.8.1 Pipeline

1. Break ML system into modules
 2. Example:
 - Image -> Text detection -> Character segmentation -> Character recognition
 - Text Detection:
 - Sliding window detection:
 - set different sizes of the window (mostly rectangle), for each size:
 - take a image patch
 - resize the patch into desired size
 - run ML algorithm on the small patch
 - slide the window by step_size (eventually through the image)
 - Expansion: expand the related region to create a bigger region
 - Character Segmentation:
 - 1-D sliding window
 - Character Recognition

2.8.2 Getting More Data

1. ****Artificial Data Synthesis**** - Creating New Data: Use available resource and combine them - Example (in Character Recognition): Paste different fonts in the randomly chosen backgrounds - Amplify Data Set: Introduce distortions to the original data set - Need to identify the appropriate distortion - Usually adding purely/random/meaningless noise
 - Prerequisite: - Having a low bias/high variance hypothesis is
 1. Collect/Label Data Manually - Usually a surprise to find how little time it needs to get 10,000 data - Calculate the time it needs before decide to/not to collect the data

2.8.3 Ceiling Analysis

1. Aim:
 - Decide which modules might be the best use of time to improve
2. Procedure:
 - Draw a table with 2 column (Component - Accuracy)
 - Component: the modules simulated to be perfect (100- Accuracy: the accuracy of the entire system on the test set (define by chosen evaluation matrix))

- — ****Perfect Component**** — ****Accuracy**** — — : ————— : —
 : ————— : — — none — f — — module 1 — $f + \epsilon_1$ — —
 module 1, 2 — $f + \epsilon_1 + \epsilon_2$ — — .
 .
 . — .
 .
 . — — module 1, 2, ..., n — $f + \epsilon_1 + \dots + \epsilon_n = 100\%$ —
 - =, Improving module x will gain at most ϵ_x improvement in the overall
 performance
 - Choose the module with most significant ϵ to improve