Machine Learning

Liyao Tang

December 11, 2018

Contents

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 $\int G_1G_2dx$
    - $G_1 \sim \mathcal{N}_d(x|a,A), G_2 \sim \mathcal{N}_d(x|b,B)$

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

  - $\Rightarrow$ Convolution of Gaussians $G1 * G2$
    - $G_1 \sim \mathcal{N}_d(a,A), G_2 \sim \mathcal{N}_d(b,B)$

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

Chapter 2

Introduction

2.1 General Concern

2.1.1 Types of Learning

Supervised Learning

- Overview
  - ○ training data comprises examples of input vectors with corresponding target vectors
- Regression
  - ○ output one or more continuous variable
- Classification
  - ○ assign input to one of a finite number of discrete categories

Unsupervised Learning

- Overview
  - ○ training data consists of a set of input vectors without target vectors
- Clustering
  - ○ Goal: discover groups of similar examples
- Density Estimation
  - ○ Goal: determine the distribution of data within the input space
- Dimension Reduction
  - ○ Goal: project data into low dimension, for the purpose of such as visualization

Reinforcement Learning

- Overview
  - ○ input with time series & discover optimal output by a process of trial and error
- Goal
  - ○ find actions to take under given circumstance to maximize a reward

2.2 Probability Theory

2.3 Model Selection

2.3.1 Over-/Under-fitting

Over-fitting

- Symptom
  - good performance on training set & poor generalization
  - ⇒ good at fitting training set; bad at representing modeling underlying data source
- Cause
  - too much representation ability (to fit even the noise)
  - directly model the likelihood instead of posterior
- Remedy
  - larger dataset
  - regularization (model the posterior by accounting prior)

Under-fitting

- bad at fitting training examples & modeling underlying data source
- lack of representation ability
- ⇒ poor performance on training set & good generalization (though meaningless)

2.4 Supervised Learning

- Feature normalization: $\forall x_{ij} \in X, x_{ij} = \frac{x_{ij}-\mu_j}{\sigma_j^2}$, $X : [instance][feature]$, without $[1...1]^T$ in 1st column $X = [x_1, x_2, ..., x_m]$, m instances in total
- Regularization: add penalty for θ being large into cost function
- $J(\theta) = ... + \dfrac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$, **bias θ_0 shouldn't be penalized**

2.5 Linear Regression

- Notation
  - t: observed data
  - $y(\mathbf{x}, \mathbf{w}) = \sum_{i=0}^{M}\phi_i(\mathbf{x})w_i = \mathbf{w}^T\phi(\mathbf{x})$: model generating ground truth, with
    - $\mathbf{w}$: weight vector
    - $\phi(\mathbf{x})$: basis function for feature vector $\mathbf{x}$, with usually $\phi_0(\mathbf{x}) = 1$ as bias
- Assumption
  - Deterministic Model with Observation Noise

- $t = y(x, w) + \epsilon$, where
  $\epsilon \sim \mathcal{N}(0, \beta^{-1})$ is Gaussian noise where precision (inverse variance) β
- $\Rightarrow$ consequence
  1. likelihood $p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1})$
  2. $\mathbb{E}[t|\mathbf{x}] = \int t \cdot p(t|\mathbf{x})dt = y(\mathbf{x}, \mathbf{w})$
  3. unimodal distribution $p(t|\mathbf{x}) \Rightarrow$ extended by conditional mixture model

- Joint Likelihood
  - $P(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^T\phi(\mathbf{x}_n), \beta^{-1})$, where
    - $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}, \mathbf{t} = \{t_1, ..., t_N\}$
  - Log Likelihood
    - $\ln P(\boldsymbol{y}|X, \theta, \beta) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta\frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$

- Log Posterior leads to regularization
  - Maximizing the likelihood function $\Rightarrow$ (often) excessively complex models & overfitting
  - 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 :
      $\Rightarrow \ln p(\theta|X) \propto -\frac{\beta}{2}\sum_{i=1}^{n}(y^i - h_\theta(x))^2 - \frac{\alpha}{2}\theta^T\theta + const$
    - If $\alpha \to 0$ (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 Guassian noise
    $p(\theta|x, X, Y) = p(\theta|X, Y) =$ posterior
  - $\Rightarrow p(y|x, X, Y) = \int p(y|\theta, x)p(\theta|X, Y)d\theta$
  - Expected Lost $= (bias)^2 + variance + noise$

- Notation:
  - $t = y(x, w) + \epsilon,$ where ϵ is Gaussian noise
  - $\hat{y}$ is prediction function to approximate $y = y(x, w)$

- Procedure:

- ○ $\mathbb{E}[(t-\hat{y})^2] = \mathbb{E}[t^2 - 2t\hat{y} + \hat{y}^2]$
  $= \mathbb{E}[t^2] + \mathbb{E}[\hat{y}^2] - \mathbb{E}[2t\hat{y}]$
  $= \mathrm{Var}[t] + \mathbb{E}[t]^2 + \mathrm{Var}[\hat{y}] + \mathbb{E}[\hat{y}]^2 - 2y\mathbb{E}[\hat{y}]$
  $= \mathrm{Var}[t] + \mathrm{Var}[\hat{y}] + (y^2 - 2y\mathbb{E}[\hat{y}] + \mathbb{E}[\hat{y}]^2)$
  $= \mathrm{Var}[t] + \mathrm{Var}[\hat{y}] + (y - \mathbb{E}[\hat{y}])^2$
  $= \mathrm{Var}[t] + \mathrm{Var}[\hat{y}] + \mathbb{E}[t-\hat{y}]^2$
  $= \sigma^2 + \mathrm{Var}[\hat{y}] + \mathrm{Bias}[\hat{y}]^2$
  where $\sigma^2 = \mathrm{Var}[\epsilon]$ is the noise
  (formula used: $\mathrm{Var}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 \Leftrightarrow \mathbb{E}[x^2] = \mathrm{Var}[x] + \mathbb{E}[x]^2$)
- ○ Matrix inverse can be evil & avoid inverse operation:
  $A = U\Lambda U^T$, where Λ is diagonal matirx
  => $A^{-1} = U\Lambda^{-1}U^T$
  but number on the diagonal line of Λ can be small =¿ maybe 0 depending on accuracy of computer

2.6 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 : $p(\boldsymbol{t}|w) = \prod_{n=1}^{N} \mathcal{N}(t_n|w^T\phi(x_n), \beta^{-1}) = \mathcal{N}(\boldsymbol{t}|\Phi w, \beta^{-1}I)$
- $\Rightarrow$ Posterior : $p(w|\boldsymbol{t}) = \mathcal{N}(w|m_N, S_N)$
  where $m_N = S_N(S_0^{-1}m_0 + \beta\Phi^T\boldsymbol{t}), S_N^{-1} = S_0^{-1} + \beta\Phi^T\Phi$
- Maximum Likelihood:
  - ○ Likelihood : $p(\boldsymbol{t}|w) = \prod_{n=1}^{N} \mathcal{N}(t_n|\phi(x_n)w, \beta^{-1})$
    - ▪ meaning: how probable the observed dataset is w.r.t the model setting (under parameter w)
  - ○ $\ln \mathrm{Likelihood} = \sum_{n=1}^{N}[-\ln\frac{\beta}{\sqrt{2\pi}} - \frac{\beta}{2}(t_n - \phi(x)w)^2]$
  - ○ $\frac{\partial}{\partial w}\ln \mathrm{Likelihood} = \beta\Phi^T(\boldsymbol{t} - \Phi w)$
    let $\frac{\partial}{\partial w}\ln \mathrm{Likelihood} = 0$
    $\Rightarrow w_{ML} = (\Phi^T\Phi)^{-1}\Phi^T\boldsymbol{t}$
  - ○ $\frac{\partial}{\partial \beta}\ln \mathrm{Likelihood} = -N\beta^{\frac{1}{2}} + \beta^{\frac{3}{2}}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w)$
    let $\frac{\partial}{\partial \beta}\ln \mathrm{Likelihood} = 0$
    $\Rightarrow \beta^{-1} = \frac{1}{N}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w)$
    Note: solve $w = w_{ML}$ first
- Maximum Posterior:

  - $\text{Posterior} = p(w|\boldsymbol{t}), \text{Prior} = p(w), \text{Likelihood} = p(\boldsymbol{t}|w), \text{Normalization} = p(t)$
    $\Rightarrow \text{Posterior} = \dfrac{\text{Likelihood*Prior}}{\text{Normalization}}$
    $\Rightarrow \text{Posterior} \propto \text{Likelihood*Prior}$
  - assume Prior $p(w) = \mathcal{N}(w|m_0, S_0)$,
    so that Prior & Likelihood are conjugate ⇒ Gaussian Posterior
  - Likelihood $p(\boldsymbol{t}|w) = \prod_{n=1}^{N} \mathcal{N}(t_n|\phi(x_n)w, \beta^{-1}) = \mathcal{N}(\boldsymbol{t}|\Phi w, \beta^{-1}I)$
  - $\Rightarrow$ Posterior $p(w|\boldsymbol{t}) = \mathcal{N}(w|m_N, S_N)$,
    where $m_N = S_N(S_0^{-1}m_0 + \beta\Phi^T\boldsymbol{t}), S_N^{-1} = S_0^{-1} + \beta\Phi^T\Phi$
    $\Rightarrow w_{MAP}$ = 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 $p(w|\boldsymbol{t}) = \mathcal{N}(w|m_N, S_N)$,
  where $m_N = \beta(\alpha I + \beta\Phi^T\Phi)\Phi^T\boldsymbol{t}, S_N^{-1} = \alpha I + \beta\Phi^T\Phi$

  $w_{MAP} \to w_{ML},$ when $\alpha \to 0$ (most useless Prior)

- Maximize Posterior ⇔ Minimize cost function with regularization:

  Simple Prior $\Rightarrow \ln p(w|\boldsymbol{t}) = -\dfrac{\beta}{2}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w) - \dfrac{\alpha}{2}w^Tw + const$

- If $\alpha \to 0$ (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 : $p(x|\alpha) = \mathcal{N}(x|0, \alpha^{-1}I), \ (m_0 = 0, S_0 = \alpha^{-1}I)$

- $p(t|x, X, \boldsymbol{t}) = \int p(t|w, x)p(w|X, \boldsymbol{t})dw$

- $\Rightarrow p(t|x, X, \boldsymbol{t}) = \mathcal{N}(t|m_N^T\phi(x), \sigma_N^2(x))$
  where $\sigma_N^2(x) = \frac{1}{\beta} + \phi(x)^TS_N\phi(x); m_N, S_N$ from Posterior$(m_N = w_{MAP})$

- Sequential data:

  - Posterior from previous data ⇔ the Prior for the arriving data
  - Sequential data and data in one go is equivalent when finfding the Porsterior

- Gradient descent

  - Hypothesis function:
    - $h_\theta(x) = x\theta$, $\theta = [\theta_0, \theta_1, ..., \theta_n]^T$, $x = [x_0, x_1, ..., x_n], x_0 = 1$
  - x is one instance
  - Cost function: $J(\theta) = \dfrac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 + \dfrac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$

- ○ Update rule: $\forall \theta_j \in \theta, \theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$, $\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^i) - y^i)x_j^i] + \frac{\lambda}{m}\theta_j$ - $\frac{d}{d\theta}J(\theta) = \frac{1}{m}((X\theta - y)^T X)^T + \frac{\lambda}{m}[0, \theta_1, ..., \theta_m]^T$ (θ_0 shouldn't be penalized)
- ○ simultaneously for all $\theta_j \in \theta$
- ○ Normal equation (mathematical solution)
  - ■ $\theta = (X^T X)^{-1} X^T y$

2.7 Logistic Regression (Classification)

- Decision Theory:

- classes $C_1, ..., C_K$, decision regions $\mathcal{R}_1, ..., \mathcal{R}_K$ - Minimze $p(mistake) = \sum_{k=1}^{K}(\int_{\mathcal{R}_k} \sum_{i \neq k} p(x, C_i)dx)$ (can have weight on each misclassfication though) - Maximize $p(correct) = \sum_{k=1}^{K} \int_{\mathcal{R}_k} p(x, C_k)dx$

- 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 trainning 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 $w = [w_1, ..., w_n]^T, x = [x_1, ..., x_n]$ - $y(x) > 0$: 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

$\Rightarrow k\frac{w}{\|w\|}$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 $\mathcal{C}_k$ if $\forall j \neq k, y_k(x) > y_j(x)$

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

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

$\Rightarrow$ optimal $W = (X^TX)^{-1}X^TT$
note that $tr\{AB\} = A^TB^T$
2. Fisher's Linear Discriminant
- Basic idea:
- Take linear classification $y = w^Tx$ 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: $\widetilde{m}_k = w^Tm_k$, where $m_k = \frac{1}{N_k}\sum_{x\in C_k} x$ - Variance: $\widetilde{s}_k = \sum_{x\in C_k}(w^Tx - w^Tm_k)^2 = w^T[\sum_{x\in C_k}(x-m_k)(x-m_k)^T]w$
- 2-Classes to 1-D line:
- Maximize Fisher criterion: $J(w) = \frac{|\widetilde{m}_1 - \widetilde{m}_2|^2}{\widetilde{s}_1^2 + \widetilde{s}_2^2}$
- Between-class covariance: $S_B = (m_1 - m_2)(m_1 - m_2)^T$
- Within-class covariance: $S_k = \sum_{n\in C_k}(x_n - m_k)(x_n - m_k)^T$
$\Rightarrow J(w) = \frac{w^TS_Bw}{w^TS_Ww}$
- Lagrangian: $L(w,\lambda) = w^TS_Bw + \lambda(1 - w^TS_Ww)$
fix w^TS_Ww to 1 to avoid infinite solution (any multiple of a solution is a solution)
$\Rightarrow \frac{\partial}{\partial w}L = 2S_Bw - 2\lambda S_Ww = 0$
$\Rightarrow S_Bw = \lambda S_Ww$
$\Rightarrow ({S_W}^{-1}S_B)w = \lambda w$
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: $J(w) = \frac{tr\{W^TS_BW\}}{tr\{W^TS_WW\}}$
- Lagrangian:
Solve for each $w_i \in W \Rightarrow ({S_W}^{-1}S_B)w_i = \lambda_iw_i$
$\Rightarrow W$ conosists 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 = $K - 1$:
$r({S_W}^{-1}S_B) \leq \min(r({S_W}^{-1}), r(S_B)) \leq r(S_B)$
$r(S_B) \leq \sum_K r((m_k - m)(m_k - m)^T) = K$, as $r(m_k - m) = 1$
$\sum_K m_k = m \Rightarrow r(m_1 - m, ..., m_K - m) = K - 1$

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

3. Perceptron Algorithm

- Generalised linear model $y = f(w^T\phi(x))$, where $\phi(x)$ is basis function; $\phi_0(x) = 1$

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

- 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 traning pair (x_n, t_n)

update the weight vector w: $w = w - \eta\nabla E_p(w) = w + \phi_n t_n$

where η=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 $\boldsymbol{X}, \boldsymbol{t}$, N is the num of data - $p(\boldsymbol{X}, \boldsymbol{t}|\pi, \mu_1, \mu_2, \Sigma) = \prod_{n=1}^{N}[\pi\mathcal{N}(x_n|\mu_1, \Sigma)]^{t_n}[(1-\pi)\mathcal{N}(x_n|\mu_2, \Sigma)^{1-t_n}] \rightarrow$ 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 - $\mu_1 = \frac{1}{N_1}\sum_{n=1}^{N}t_n x_n, \mu_2 = \frac{1}{N_2}\sum_{n=1}^{N}(1-t_n)x_n$, (mean of each class) - $\Sigma = \frac{N_1}{N}S_1 + \frac{N_2}{N}S_2$, 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: $\sigma(a) = \frac{1}{1+e^{-a}}$

- $p(x|C_k) \sim \mathcal{N} \implies p(C_k|x) = \sigma(w^Tx + w_0)$ (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

-

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

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

- $\implies p(C_1|x) = \sigma(w^Tx + w_0)$
⇒ Find parameters in Gaussian model using Maximal Likelihood Sulotion
as: $p(C_1|x) \propto p(x|C_1)p(C_1) = p(x, C_1)$
- Generalize to K-classes:
$a_k(x) = \ln[p(x|C_k)p(C_k)], p(C_k|x) = \frac{\exp(a_k)}{\sum_i \exp(a_i)}$
⇒ $a_k(x) = w_k^Tx + w_{k0}$, where $w_k = \Sigma^{-1}\mu_k; w_{k0} = -\frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + p(C_k)$
- Assume directly $p(C_k|x) = \sigma(w^Tx + w_0)$ (2-classes) (Discriminative model)
- Assume directly: $p(C_1|w, x) = \sigma(w^Tx), x_0 = 1$
⇒ less parameters to fit (compared to Gaussian)
- Likelihood function:
$p(\boldsymbol{t}|w, X) = \prod_{n=1}^{N} p_n^{t_n}(1-p_n)^{1-t_n}$, where, $p_n = p(C_1|x_n), t_n$ is the class of x_n
Define error function :
$E(w) = -\ln(Likelihood) = -\sum_{n=1}^{N}[t_n \ln p_n + (1-t_n)\ln(1-p_n)]$
⇒ $\nabla E(w) = \sum_{n=1}^{N}(p_n - t_n)x_n$
- Find Posterior $p(w|\boldsymbol{t})$:
Likelihood is product of sigmoid
Conjugate Prior for "sigmoid distribution" is unknown
⇒ Assume Prior $p(w) = \mathcal{N}(w|m_0, S_0)$
⇒ $\ln p(w|\boldsymbol{t}) \propto -\frac{1}{2}(w - m_0)^T S_0^{-1}(w - m_0) + \sum_{n=1}^{N}[t_n \ln p_n + (1-t_n)\ln(1-p_n)]$
find w_{MAP}, calculate $S_N = -\nabla\nabla \ln p(w|\boldsymbol{t}) = S_o^{-1} + \sum_{n=1}^{N} p_n(1-p_n)\phi_n\phi_n^T$
⇒ $p(w|\boldsymbol{t}) \simeq \mathcal{N}(w|w_{MAP}, S_N)$, via Laplace Approximation
- Laplace Approximation:
- Fit a guassian 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: $f(z) \simeq f(z_0)\text{exp}-\frac{A}{2}(z-z_0)^2$
$\Rightarrow$ Laplace Approximation $= (\dfrac{A}{2\pi})^{1/2}\text{exp}-\dfrac{A}{2}(z-z_0)^2$, where $A = \dfrac{1}{\sigma^2}$
- 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 $A = -\nabla\nabla \ln f(z)|_{z=z_0}$

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

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

- Gradient descent:
- Hypothesis function: $h_\theta(x) = \sigma(x\theta) = \frac{1}{1+e^{-x\theta}}$
- Cost function:
$J(\theta) = \dfrac{1}{m}\sum_{i=1}^{m}[-y^i \ln(h_\theta(x^i)) - (1-y^i)\ln(1-h_\theta(x^i))] + \dfrac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$
- Update rule: $\forall\theta_j \in \theta, \theta_j := \theta_j - \alpha\frac{\partial J(\theta)}{\partial\theta_j}$, $\dfrac{\partial J(\theta)}{\partial\theta_j} = \dfrac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^i)-y^i)x_j^i] + \dfrac{\lambda}{m}\theta_j$

2.8 Diagnose Machine Learning

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

2.8.1 Evaluating Hypothesis

1. Data set =¿ training set (60
- randomly split
2. Cross Validation:
- estimation of the generalization error, inaccurate because:
- finite trainning set - finite corss validation set
- S-fold Cross Validation:
- devide 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 traning set
run the procedure (mentioned later) on the traning set
estimate generalization in CV set
3. Test set error
- Linear regression: $J_{test}(\theta) = \dfrac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_\theta(x^i_{test})-y^i_{test})^2$ - Logistic regression: $J_{test}(\theta) = -\dfrac{1}{m_{test}}\sum_{i=1}^{m_{test}}(y^i_{test} logh_\theta(x^i_{test})+(1-y^i_{test})logh_\theta(x^i_{test}))$ - Misclassfication error: $J_{test}(\theta) = \dfrac{1}{m_{test}}\sum_{i=1}^{m_{test}} err(h_\theta(x^i_{test}), y^i_{test})$
-

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

4. Choosing procedure:

- Minimize traning error $J_{train}(\theta)$ - Select a model with lowest $J_{cv}(\theta)$ - Estimate generalization error as $J_{test}(\theta)$

2.8.2 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 $\lambda = 0, 0.01, 0.02, 0.04, ..., 10$ - 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 polunomial terms of features - decreasing λ

- High variance:

- larger data set - fewer features - increasing λ

- **In neural network:**

- High bias =¿ larger neural networks (more hidden layers / more units in one layer)

- High variance =¿ smaller neural networks

Larger network with regularization (λ) is more powerful

2.8.3 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 =¿ Numerical evaluation - =¿ test algorithm with / without this feature on **CV set** (compare error rate)

2.8.4 Skewed classes

1. Precision / Recall

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

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

- **Recall** $= \frac{\text{True positive}}{\text{Actual positive}} = \frac{\text{True positive}}{\text{True pos + False neg}}$

2. Evaluation with precision/recall

- Predict 1 if $h_\theta(x) \geq \epsilon$, 0 if $h_\theta(x) < \epsilon$

- larger ϵ =¿ higher precision, lower recall (more confident) - smaller ϵ =¿ lower pecision, higher recall (avoid missing)

![Posiible Precision -Recall curev](../../Machine Learning/Statistical 0Machine Learning/Posiible Precision -Recall curev.png)

3. Compare precision/recall num

- F$_1$$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)**

2.9 Latent Variable Analysis

2.9.1 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:
⇒ variance of projected data is maximized
⇒ 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:
- $U = (u_1, ..., u_M)$, where $\forall i \in \{1, .., M\}, u_i \in \mathbb{R}^D$ 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 $= tr\{U^T S U\}$, where $S = \sum_{i=1}^{N}(x^i - \bar{x})(x^i - \bar{x})^T$ (outer product)
- Lagrangian to maximize Variance:
- $L(U, \lambda) = tr\{U^T S U\} + tr\{(I - U^T U)\lambda\}$
constraint $u_i^T u_i = 1$ to prevent $u_i \to +\infty$

$$\text{For each } u_i \in U,\ \frac{\partial}{\partial u_i} L = 2Su_i - 2\lambda_i u_i = 0 \tag{2.4}$$

$$\Rightarrow Su_i = \lambda_i u_i \tag{2.5}$$

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

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

- Let $\begin{cases} \dfrac{\partial}{\partial z_j^n} J = 0 \\ \dfrac{\partial}{\partial b_j} J = 0 \end{cases} \Rightarrow \begin{cases} \dfrac{1}{N} 2(x^n - \widetilde{x^n})^T(-u_j) = \dfrac{2}{N}(z_j - (x^n)^T u_j) = 0 \\ \dfrac{1}{N}\displaystyle\sum_{n=1}^{N} 2(x^n - \widetilde{x^n})^T(-u_j) = \dfrac{2}{N}\displaystyle\sum_{n=1}^{N}(b_j - (x^n)^T u_j) = 0 \end{cases}$

$\Rightarrow \begin{cases} z_j = (x^n)^T u_j & j \in \{1, ..., M\} \\ b_j = \overline{x}^T u_j & j \in \{M+1, ..., D\} \end{cases}$

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

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

-

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

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

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

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

$$\tag{2.11}$$

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

- Lagrangian to Minimize J:

- $L(u_{M+1}, ..., u_D, \lambda_{M+1}, ..., \lambda_D) = \sum_{i=M+1}^{D} u_i^T S u_i + \sum_{i=M_1}^{D} \lambda_i (1 - u_i^T u_i)$

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

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

$\Rightarrow Su_i = \lambda_i u_i$

$\Rightarrow$ To minmize 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

-

$$\text{Intuition: } \widetilde{x_n} = \sum_{i=1}^{M}((x^n)^T u_i) u_i + \sum_{i=M+!}^{D}(\overline{x}^T u_i) u_i \tag{2.12}$$

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

1. Singular Value Decomposition - SVD:

- Intorduce matrix $A_{m \times n}$

- $(A^TA)_{n\times n}$ symmetric matrix (actually, Gram matrix → semi-definite) -

$$\text{eigenvalue decomposition:} \tag{2.14}$$

$$A^TA = VDV^T,\ V \text{ is normalized } (v_i^Tv_i = 1) \text{ with column as eigenvector} \tag{2.15}$$

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

$$\Rightarrow r(A^TA) = r(A) = r \tag{2.16}$$

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

- Reduce AV to basis $(Av_1, ..., Av_r)$
- Let $U = (u_1, ..., u_r) = (\dfrac{Av_1}{\sqrt{\lambda_1}}, ..., \dfrac{Av_r}{\sqrt{\lambda_r}})$, λ_i is i-thh eigenvalue of A^TA
- Orthogonal: $\forall i \neq j, u_i^Tu_j = \frac{1}{\sqrt{\lambda_i\lambda_j}}v_i^TA^TAv_j = \frac{\lambda_j}{\sqrt{\lambda_i\lambda_j}}v_i^Tv_j = 0$
- Unit: $\|u_i\| = \frac{\|Av_i\|}{\sqrt{\lambda_i}} = \frac{\sqrt{<Av_i,Av_i>}}{\sqrt{\lambda_i}} = 1$

$\Rightarrow U$ is standard orthogonal (orthonormal) basis
- $AV = U\Sigma,$ where $\Sigma = D^{\frac{1}{2}}$
- Expand U to orthonormal in $\mathbb{R}^m : (u_i, ..., u_m)$
- Epand 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 corvariance matrix $S = X^TX$ 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
- $S = VDV^T$ with V orthonormal:

Eigenvalues λ of covariance matrix S express the variance of data set X in direction of corresponding eigenvectors
- Projection:
- $\widetilde{X} = XV_M,$ 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*k matrix, z is k*1 vector - ![Reconsturction from data Compression](../../Machine

4. Choosing k (num of principal components):
- choose the **smallest** k making $\dfrac{J}{V} \leq 0.01$ =¿ 99
- $[U, S, V] =$ svd(Sigma) =¿ $\dfrac{J}{V} = 1 - \dfrac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}}$, S is diagonal matrix

=¿ 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 - $\forall n, y_n = D^{-\frac{1}{2}}V^T(x^n - \overline{x})$, where $\overline{x}$ is the mean of X

$$\Rightarrow y^n \text{ has zero mean} \tag{2.18}$$

$$cov(\{y^n\}) = \frac{1}{N}\sum_{n=1}^{N} y_ny_n^T = D^{\frac{-1}{2}}V^TSVD^{\frac{-1}{2}} = I \tag{2.19}$$

6. Tips for PCA:

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

2.9.2 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 - Ignorde 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

2.9.3 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 ¡u¿local distances¡/u¿ in the original data

2.9.4 Anomaly Detection

1. Problem to solve:

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

2. Hypothesis function:

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

3. Multivariate Gaussian:

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

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

- $\mu = \frac{1}{m}\sum_{i=1}^{m} x^i, \Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^i - \mu)(x^i - \mu)^T$ - $x_1, ...x_n$ can be correlated but **not** linearly dependent - need $m > n$ ($m \geq 10 nsuggested$) or elas 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-guassian feature: transformation / using other distribution - Choosing features: compare anomaly data with normal data

2.9.5 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

=¿ for user j, movie i, $(r_{i,j} = 0)$, predict rating $x^i\theta^j$

2. Content Based Recommendations:

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

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

=¿ given X estimate Θ

- Cost Function for θ_j:

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

- Cost Function for Θ:

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

$\theta^j \in R^{n+1}(\theta_0 \text{ not regularized}), n_u$ is num of users

- Update Rule: $\forall\theta^j_k \in \theta^j, \theta^j_k := \theta^j_k - \alpha\frac{\partial J(\Theta)}{\partial \theta^j_k}, \dfrac{\partial J(\Theta)}{\partial \theta^j_k} = \sum_{i:r_{i,j}=1}(x^i\theta^j - y_{i,j})x^i_k + \lambda\theta^j_k$, 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: $J(x^i) = \dfrac{1}{2}\sum_{j:r_{i,j}=1}(x^i\theta^j - y_{i,j})^2 + \dfrac{\lambda}{2}\sum_{k=1}^{n}(x^i_k)^2$ - Cost Function for X:

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

$x^j \in R^{n+1}(x_0 \text{ not regularized}), n_m$ is num of items - Update Rule: $\forall x^i_k \in x^i, x^i_k := x^i_k - \alpha\frac{\partial J(X)}{\partial x^i_k}$, $\dfrac{\partial J(X)}{\partial x^i_k} = \sum_{j:r_{i,j}=1}(\theta^j x^i - y_{i,j})\theta^j_k + \lambda x^i_k$, for $k \neq 0$ $(x^i \in R^{n+1})$

- Basic Idea:

- Randomly initialize Θ

- loop:

Estimate X

Estimate Θ

- Cost Function:

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

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

- Update Rule:

- $\forall x^i_k \in x^i, x^i_k := x^i_k - \alpha \frac{\partial J(X,\Theta)}{\partial x^i_k}, \dfrac{\partial J(X,\Theta)}{\partial x^i_k} = \dfrac{\partial J(X)}{\partial x^i_k} = \sum_{j:r_{i,j}=1} (\theta^j x^i - y_{i,j})\theta^j_k + \lambda x^i_k, x^i \in R^n$

- $\forall \theta^j_k \in \theta^j, \theta^j_k := \theta^j_k - \alpha \frac{\partial J(X,\Theta)}{\partial \theta^j_k}, \dfrac{\partial J(X,\Theta)}{\partial \theta^j_k} = \dfrac{\partial J(\Theta)}{\partial \theta^j_k} = \sum_{i:r_{i,j}=1} (\theta^j x^i - y_{i,j})x^i_k + \lambda \theta^j_k, \theta^j \in R^n$

- **Algorithm**
- Initialize X, Θ to **small random values**
=¿ for symmetry breaking (similar to random initialization in neural network)
=¿ so that algorithm learns features $x^1, ..., x^{n_m}$ 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 samll =¿ item i and j is similar
- Mean Normalization:
- Problem: if user j hasn't rated any movie, $\theta^j = [0, ..., 0]$
=¿ predicted rating of user j on all item = 0
=¿ useless prediction
- Algorithm (row version):
compute vector $\mu, \forall \mu_i \in \mu, \mu_i =$ mean of Y_i, where Y_i is the i^{th} row in Y
manipulate Y: $\forall y_{i,j} \in Y \wedge r_{i,j} = 1, y_{i,j} -= \mu_i$ =¿ 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)

2.10 Large Scale Machine Learning

2.10.1 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: $J_{train}(\theta) = \dfrac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^i, y^i))$
- Procedure:
- Randomly shuffle dataset
- Repeat
for $i \in [1, m]$

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

$$(\text{for } j = 0, ..., n) \tag{2.21}$$

$$\text{=> make progress with each single data} \tag{2.22}$$

- Convergence:
- Wanting θ to converge =¿ slowly decrease α over time (but more parameters)
(E.g $\alpha = \dfrac{\text{const\_1}}{\text{iteration num + const\_2}}$)
- Compute $cost(\theta, (x^i, y^i))$ before updating
For every k update iterations, plot average $cost(\theta, (x^i, y^i))$ 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 =¿ smaller α
- *vs* Batch Gradient Descent:
- use 1 example un each update iteration =¿ make progress earlier =¿ 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 =¿ faster - Result may not be the optimal but in its neighbourhood - *vs* Stochatistic Gradient Descent:t - can partially parallelize computation over b examples =¿ faster under a good vectorized implementation & appropriate b - introduce extra parameter b

2.10.2 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

2.10.3 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)

2.11 Building Machine Learning System

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

2.11.1 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
- Chraracter Segmentation:
- 1-D sliding window
- Character Recognition

2.11.2 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: Intorduce distortions to the original data set - Need to identify the appropriate distortion - Usually adding purely/random/meanless 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 - Caculate the time it needs before decide to/not to collect the data

2.11.3 Ceilling 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 $1, 2, ..., n$ — $f + \epsilon_1 + ... + \epsilon_n = 100\%$ —

- =¿ Improving module x will gain at most ϵ_x improvement in the overall performance

- Choose the module with most significant ϵ to improve

Chapter 3

Linear Regression

Chapter 4

Linear Classification

Chapter 5

Kernel Methods

Chapter 6

Graphical Models

Chapter 7

Mixture Models and EM

Chapter 8

Approximate Inference

Chapter 9

Sampling Methods

Chapter 10

Continuous Latent Variable

Chapter 11

Sequential Data

Chapter 12

Deep Learning

12.1 Interview of Fame

12.1.1 Geoffrey Hinton

Knowledge Embedding

- BP
    - psychology view: knowledge in vectors
    - semantic AI: knowledge graph
    - BP algorithm can interpret & convert between feature vector and graph representation (with some embedding)
- Boltzmann Machine
    - Leaning Algorithm on Density Net
        - same information in forward & backward propagation to learn feature embedding
    - Restricted Boltzmann Machine (RBM)
        - ways of learning in deep dense net with fast inference
        - iterative learning (adding layer after the above trained)
        - ReLU ⇔ a stack of sigmoid functions (approximately) in RBM
        - ReLU units initialized to identity for efficient learning
- EM
    - EM with Approximate E Step
- vs. Symbolic AI
    - Symbolic AI: symbolic logic-like expression to do reasoning
    - yet, maybe state vector to represent knowledge

Brain Science

- Brain: Nets Implemented by Evolution
    - trying to train without BP
    - doing BP (get derivatives) with re-construction error (auto-encoder)

Memory in Nets

- Fast Weights for Short-term Memory
- Capsule Net
  - structured knowledge representation in each unit (feature with sets of property)
  - ⇒ enable nets to vote rather than filtering - thus better generalization

Unsupervised Learning

- Importance
  - better than human eventually (as supervised learning has limited maximum)
  - GAN as a breakthrough

"Slow" Feature

- Non-linear Transform to Find Linear Transform
  - find a latent representation containing linear transform to do the work
  - e.g. change viewpoints: pixels → coordinates → linear transform → back to pixels

Relations between Computers

- showing computer data to work
  - instead of programming it to work

12.1.2 Pieter Abbeel

Deep Reinforcement Learning

- Overall Challenge
  - Representation
  - Exploration Problem
  - Credit Assignment
  - Worst Case Performance
- Advantage (Deep Nets in RL)
  - network capturing the representation (state vector)
- Question in DRL
  - how to learn safely
  - how to keep learning (under small negative samples) e.g. better than human
  - can we learn the reinforcement learning program (RL in the RL)
  - long time horizon
  - use experience across tasks
- Success of DRL
  - simulated robot inventing walking... ⇒ single general algorithms to learn

12.1.3 Research Advices

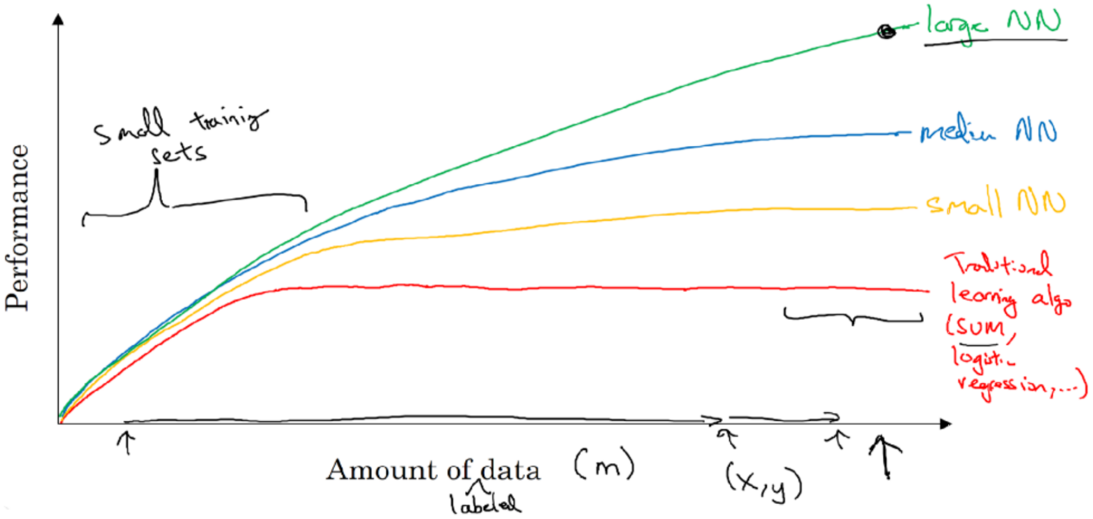
- Reading
  - read a little bit & find somewhere intuitively not right
    - good intuition: eventually work; bad intuition: not working no matter what it is doing
    - if other doubts your idea as bullshit ⇒ a sign for real good result
  - a supervisor with similar belief
  - PhD vs. Company
    - amount of mentoring
    - faster if dedicated supervisor available
- Practice
  - open-source learning resource
  - implement the paper

12.2 Basic Neutral Network

12.2.1 Goal and Advantages

Data Mining and Pattern Recognition

- Larger Maximum Capability
  - Curve given Amount of Data



  - Reasons
    - the scale of data (labeled)
    - the scale of neural network (computability)
    - the scale of efficiency: e.g. ReLu, faster parallel algorithm

Ability to Choose

- Choosing Basis Functions
    - Functional View
        - $y(\mathbf{x}, \mathbf{w}) = f(\mathbf{w}^T \phi(\mathbf{x})),$ where ϕ is basis function , $f(\cdot)$ is net as a function
    - Learning ϕ: choose embedding ⇒ choose basis function
    - Learning $\mathbf{w}$: choose which feature / basis functions more useful

12.2.2 Problem

(n units in one hidden layer)

Weight-space Symmetries

- Symmetries in Activation Function
    - $\mathcal{O}(2^n)$, e.g. $\arctan(-x) = -\arctan(x) \Rightarrow$ changing signs of all input & output has the same mapping (reduce effective data)
- Positional Combination in One Layer
    - $\mathcal{O}(n!)$ exchange unit with each other (together with their input output weights) ⇒ mapping stay the same

⇒ $\mathcal{O}(n!2^n)$ overall weight-space symmetries

Non-convex Error Function

- Multiple Critical Points
    - at least $\mathcal{O}(n!2^n)$ critical points ($\nabla E(w) = 0,$ where $E(w)$ is error function) due to weight-space symmetries
- Expensive in Finding Critical Point
    - expensive for even local optima with gradient decent
    - as expensive as $\mathcal{O}(n^3)$ if using Laplace approximation

Gradient Vanishing

- Ability of Depth (however)
    - functions that can be compactly represented by a depth k architecture might require an exponential number of computational nodes using a depth $k-1$ architecture
- Possible Solutions
    - Xavier initialization, ReLU activation, ...

12.2.3 Learning

Forward-Backward Propagation

- Representation
    - Layers
        - input layer, hidden layer(s) (each has a bias term $x_0 = 1$)
        - output layer
    - Neuron (Unit)
        - s_l: num of units in layer l
        - w^l: weight matrix of mapping from layer l to $l+1$, with shape of $s_{l+1}, (s_l + 1)$
        - a^l_j: activation of unit j at layer l
        - $h(\cdot)$: activation function (usually shared)
        - z^l_j: output of unit j at layer l (represent parameterized basis)
- Forward Propagation
    - Activation $a^{j+1} = w^j \cdot [z^j_0, ..., z^j_{s_j}]^T,$ with $z_0 = 1$
    - Unit Output $z^{j+1} = h(a^{j+1}) = [z^1_1, ..., z^{j+1}_{s_{j+1}}]^T$
- Backward Propagation
    -