Lecture Syllbus

cs229-notes1

cs229-notes

ML Terminology

Classification

Supervised learning

Unsupervised Learning

Parametric Learning Algorithms

Non-Parametric Learning Algorithms

Discriminative Learning algorithms

Linear Regression

normal equations

Locally weighted linear regression

Generalized Linear Models

exponential family

3 Different Ways to Parameterize GLM

Softmax Regression

Classification and logistic regression

binary classification

Logistic regression

perceptron

LMS algorithm

gradient descent

batch gradient descent

stochastic gradient descent

gradient ascent

stochastic gradient ascent

gradient ascent

Newton's method

Discriminative VS Generative Learning Algorithms

GDA vs Logistic Regression

Newton's Method vs Gradient Ascent

Supervised Learning Algorithms

Generative Learning algorithms

Gaussian discriminant analysis (GDA)

multivariate normal distribution / multivariate Gaussian distribution

Laplace Smoothing

Naive Bayes

Event Models

Multi-Variate Bernoulli Event Model

Multinomial Event Model

Support Vector Machines

Optimal Margin Classifiers

Functional Margin

Geometric Margin

Kernels

Non-Separable Case

L1 Regularization

L2 Regularization

Empirical Risk Minimization (ERM)

Hoeffding Inequality

Decision Trees

Cross Entropy Loss

Ensembling

DeepLearning

Xavier/He Initialization

Gaussian Discrimination Analysis VS Mixture of Gaussian

Algorithms

Mathematical Methods

The K-Means Clustering Algorithm

Mixture of Gaussians

Jensen's inequality

Unsupervised Learning Algorithms

Expectations-Maximization

Density Estimation

Factor Analysis

Marginals and Conditionals of Gaussians

EM for Factor Analysis

Independent Components Analysis

Reinforcement Learning and Control

Markov Decision Processes

Finite-State MDPs

Value Iteration

Synchronous Update

Asynchronous Updates

Policy Iteration

(Direct) Policy Search

Value Iteration VS Policy Iteration

Continuous State MDPs

Discretization

Value Function Approximation

Fitted Value Iteration

(Direct) Policy Search

Direct Policy Search vs Value-function-based Approach

MDP with unknown state transition probabilities

Exploration vs Exploitation problem

State-Action Rewards

Finite Horizon MDP

Linear Quadratic Regulation (LQR)

Linearize a non-linear model

Probabilistic interpretation

Posterior Distribution

Statistics

model (e.g p (x, y, θ))

Prior Distribution

Regularization

Cross Validation

Hold-Out Cross Validation (Simple Cross Validation)

K-Fold Cross Validation

Feature Selection

Modeling Strategies

Error Analysis

Ablative Analysis

cs229-ps

cs229-ps_sol

Machine Learning Yearning

机器学习 Machine Learning

图解机器学习

cs229-extra_notes

cs229-notes

- ML Terminology
  - Classification
    - Unsupervised Learning
- Mathematical Methods
  - Algorithms
- Group
- Lectures
  - Lecture 2
    - Supervised learning
    - Linear Regression
      - normal equations
      - Locally weighted linear regression
    - LMS algorithm
      - gradient descent
        - batch gradient descent
        - stochastic gradient descent
    - Probabilistic interpretation
    - Parametric Learning Algorithms
    - Non-Parametric Learning Algorithms
  - Lecture 3
    - Classification and logistic regression
      - binary classification
      - Logistic regression
        - gradient ascent
          - gradient ascent
          - stochastic gradient ascent
        - Newton's method
    - Newton's Method vs Gradient Ascent
  - Lecture 4
    - perceptron
    - exponential family
    - Generalized Linear Models
    - Softmax Regression
  - Lecture 5
    - Generative Learning algorithms
      - Gaussian discriminant analysis (GDA)
      - Naive Bayes
        - Laplace Smoothing
        - Event Models
          - Multi-Variate Bernoulli Event Model
          - Multinomial Event Model
  - Lecture 6 Supportive Vector Machine
    - Support Vector Machines
      - Optimal Margin Classifiers
        - Functional Margin
        - Geometric Margin
      - Kernels
      - Non-Separable Case
        - L1 Regularization
        - L2 Regularization
  - Lecture 7 Kernels
  - Lecture 8
    - Regularization
      - Cross Validation
        - Hold-Out Cross Validation (Simple Cross Validation)
        - K-Fold Cross Validation
      - Feature Selection
  - Lecture 9 Learning Theory
    - Approx/Estimation Error &ERM
      - Empirical Risk Minimization (ERM)
        - Hoeffding Inequality
  - Lecture 10 Decision Trees & Ensembling
    - Decision Trees
      - Cross Entropy Loss
    - Ensembling
  - Lecture 11 Introduction to Neural Networks
    - DeepLearning
      - Xavier/He Initialization
  - Lecture 12 Backpropagation & Improving Neural Networks
  - Lecture 13 Debugging ML Models & Error Analysis
    - Error Analysis
    - Ablative Analysis
  - Lecture 14 EM algorithms
    - Expectations-Maximization
      - The K-Means Clustering Algorithm
      - Mixture of Gaussians
      - Jensen's inequality
      - Density Estimation
  - Lecture 15 Factor Analysis
    - Factor Analysis
      - Marginals and Conditionals of Gaussians
      - EM for Factor Analysis
  - Lecture 16 Independent Components Analysis & Reinforced Learning
    - Independent Components Analysis
    - Reinforcement Learning and Control
      - Markov Decision Processes
        - MDP with unknown state transition probabilities
          - Exploration vs Exploitation problem
        - State-Action Rewards
        - Finite Horizon MDP
        - Linear Quadratic Regulation (LQR)
          - Linearize a non-linear model
  - Lecture 17 MDPs & Value/Ploicy Iteration
    - Finite-State MDPs
      - Value Iteration
        - Synchronous Update
        - Asynchronous Updates
      - Policy Iteration
  - Lecture 18 Continuous State MDP & Model Simulation
    - Continuous State MDPs
      - Discretization
      - Value Function Approximation
        - Fitted Value Iteration
  - Lecture 19 Reward Model & Linear Dynamical System
  - Lecture 20 RL Debugging and Diagnostics
    - RL debugging/ diagnostics
    - (Direct) Policy Search
    - Direct Policy Search vs Value-function-based Approach

- Lecture Syllbus
- cs229-notes1 1

- cs229-notes
  - ML Terminology
    - Classification
      - Supervised learning 1

        Linear Regression
      - Unsupervised Learning
      - Parametric Learning Algorithms

        Linear Regression

        Fit some fixed of parameters (**θ_i**) to data
      - Non-Parametric Learning Algorithms

        Locally weighted linear regression

        Amount of data/parameters you need to keep grows (linearly) with the size of dataset.

        - Need to keep all the data in computer memory to make predications.
  - Mathematical Methods
    - Algorithms
      - Gaussian Discrimination Analysis VS Mixture of Gaussian

        1. **Dataset** *(supervised or unsupervised)*
           For a given dataset, GDA has example $(x^{(i)}, y^{(i)})$ where $y^{(i)}$ is observed, MG doesn't.
        2. **Probability Distribution**
           For GDA, $y^{(i)} \sim Bernoulli(\phi)$
           For MG, $z^{(i)} \sim Multinomial(\phi)$
        3. **Variance for Gaussian Distribution** Σ
           For GDA, $x^{(i)} \mid y_j^{(i)} \sim \mathcal{N}(\mu_j, \Sigma)$
           For MG, $x^{(i)} \mid z_j^{(i)} \sim \mathcal{N}(\mu_j, \Sigma_j)$

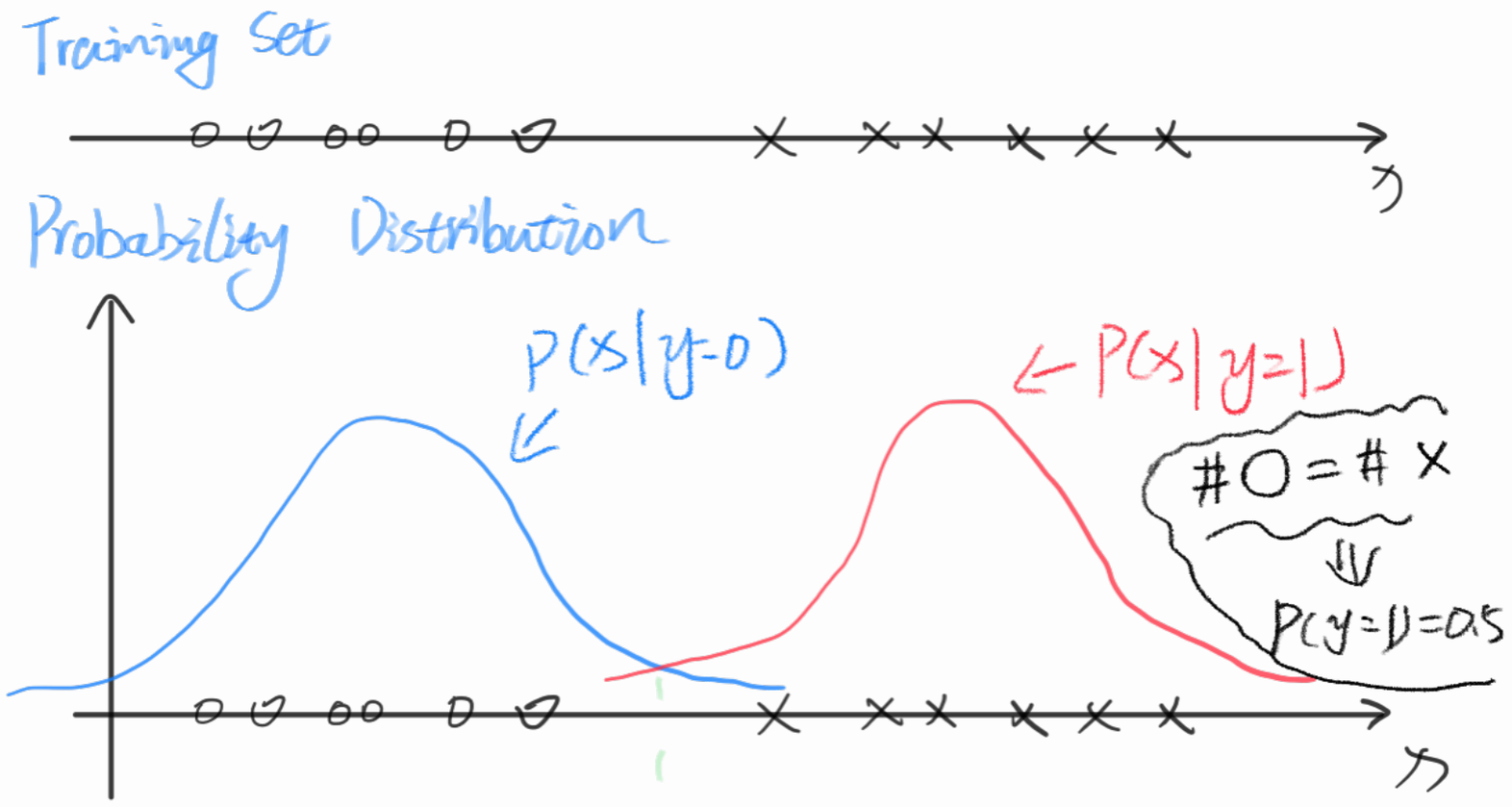
        Mixture of Gaussians

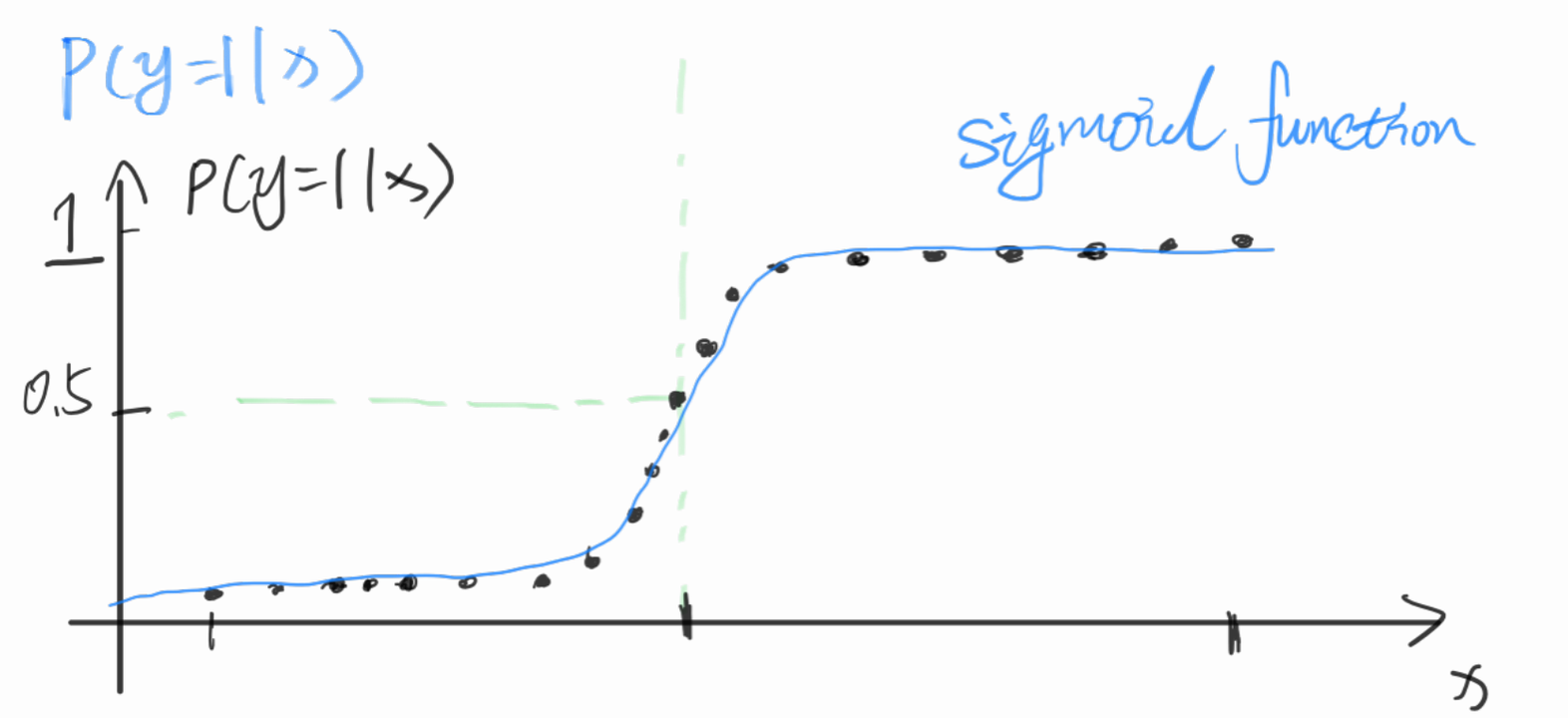
        Gaussian discriminant analysis (GDA)

- GDA vs Logistic Regression

  Gaussian discriminant analysis (GDA)

  Logistic regression





GDA model assumptions implies $p(y=1|x)$ is governed by logistic function.

(Generative) Stronger (Discriminative) Weaker

GDA assume Assumption Logistic Regression Assumption

$x|y=0 \sim N(\mu_0, \Sigma)$

$x|y=1 \sim N(\mu_1, \Sigma)$

$y \sim Ber(\phi)$

$\Rightarrow$ $P(y=1|x) = \frac{1}{1+e^{-\theta^T x}}$

$\not\Leftarrow$ $P(y=1|x)$ is logistic

Stronger assumption gives more information,

so if indeed $x|y$ is Gaussian, GDA will do better

In other word, GDA assumptions are roughly correct (especially when you have a small dataset)

Fun fact

if $x|y=0 \sim Poisson(\lambda_0)$, $x|y=1 \sim Poisson(\lambda_1)$, $y \sim Ber(\phi)$ $\Rightarrow$ $P(y=1|x)$ is logistic

For this scenario, GDA won't do great

($x|y$ isn't Gaussian)

- Discriminative VS Generative Learning Algorithms

  Discriminative Learning algorithms

  Generative Learning algorithms

- Discriminative learning algo.
  - learn $p(y|x)$
  (or learn a function mapping from x to labels directly) $x \to y$
  e.g. $h_\theta(x) = \begin{cases} 0 \\ 1 \end{cases}$
  - try to find θ to maximize $L(\theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta)$
  Conditional Likelihood

- Generative learning algo.
  - learns $P(x|y)$, $P(y)$
  feature, class, Class Prior
  - Find params to maximize Joint Likelihood
  $L(\phi,\mu_0,\mu_1,\Sigma) = \prod_{i=1}^{m} p(x^{(i)},y^{(i)};\phi,\mu_0,\mu_1,\Sigma)$
  $= \prod_{i=1}^{m} p(x^{(i)}|y^{(i)})\,p(y^{(i)})$

- Newton's Method vs Gradient Ascent

  gradient ascent

  Newton's method

  - Newton's method converges quicker than Gradient ascent.
  - While in high-dimensional (e.g. 1000, 1000000) problems, each iteration step costs more in Newton's Method.

Supervised Learning Algorithms

Discriminative Learning algorithms

Linear Regression 3

Parametric Learning Algorithms

Supervised learning

- Linear Regression has no local optimum

- normal equations 11

$$X^T X\theta = X^T\vec{y}$$

$$\theta = (X^T X)^{-1} X^T \vec{y}.$$

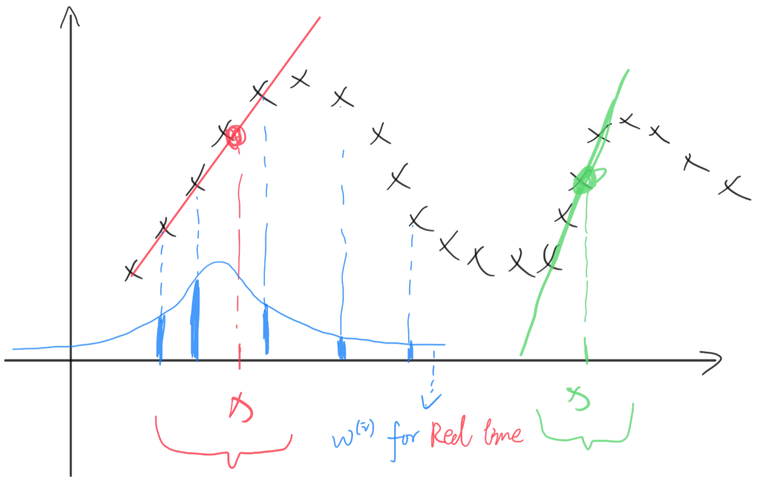
$$X^T X\theta = X^T\vec{y}$$
$$\theta = (X^T X)^{-1} X^T\vec{y}$$

#Formula

- Locally weighted linear regression 13

  Non-Parametric Learning Algorithms

  Tend to use: when you have a relatively low dimensional dataset (i.e. #features is not too big) and have a lot of data, and don't wanna think about what features to use.



1. Fit θ to minimize $\sum_i w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$.
2. Output $\theta^T x$.

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

The choice of the bandwidth parameter τ has an effect on overfitting and underfitting.
- Too big end up over-smoothing the data
- Too small end up fitting a very jagged fit to the data

Relative slow, but good enough for small dataset (up to thousands), while dataset size is too big (say, millons), use KD tree etc is better.

#Formula

Classification and logistic regression 16

- binary classification 16

- Logistic regression 16

Hypothesis

$$h_\theta(x) = g\left(\theta^T x\right) = \frac{1}{1 + e^{-\theta^T x}}$$

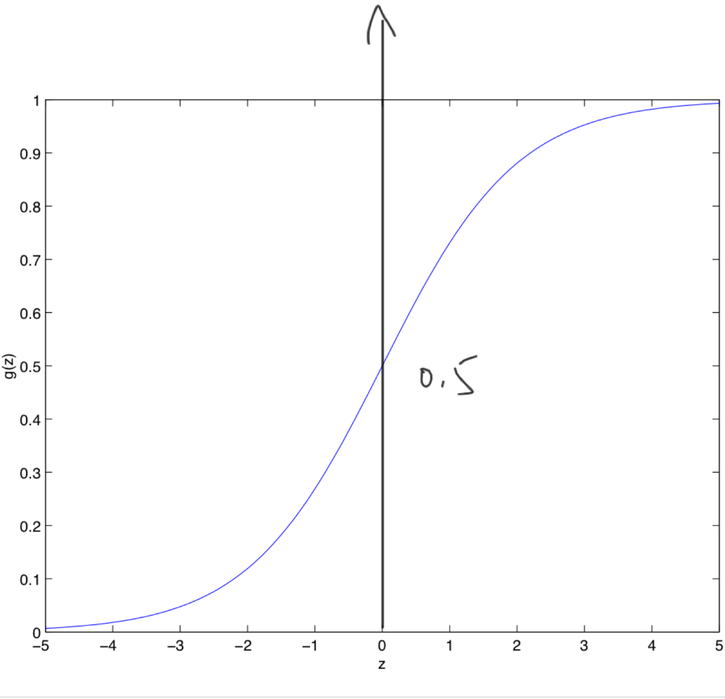
Logistic/Sigmoid Function

$$g(z) = \frac{1}{1 + e^{-z}}$$

By convention,

$$p(y \mid x; \theta) = (h_\theta(x))^y \left(1 - h_\theta(x)\right)^{1-y}$$

g(z) demenstration



UPDATE RULES

gradient ascent

Newton's method

#Formula

- perceptron

19

"Hard mode" of sigmoid function

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

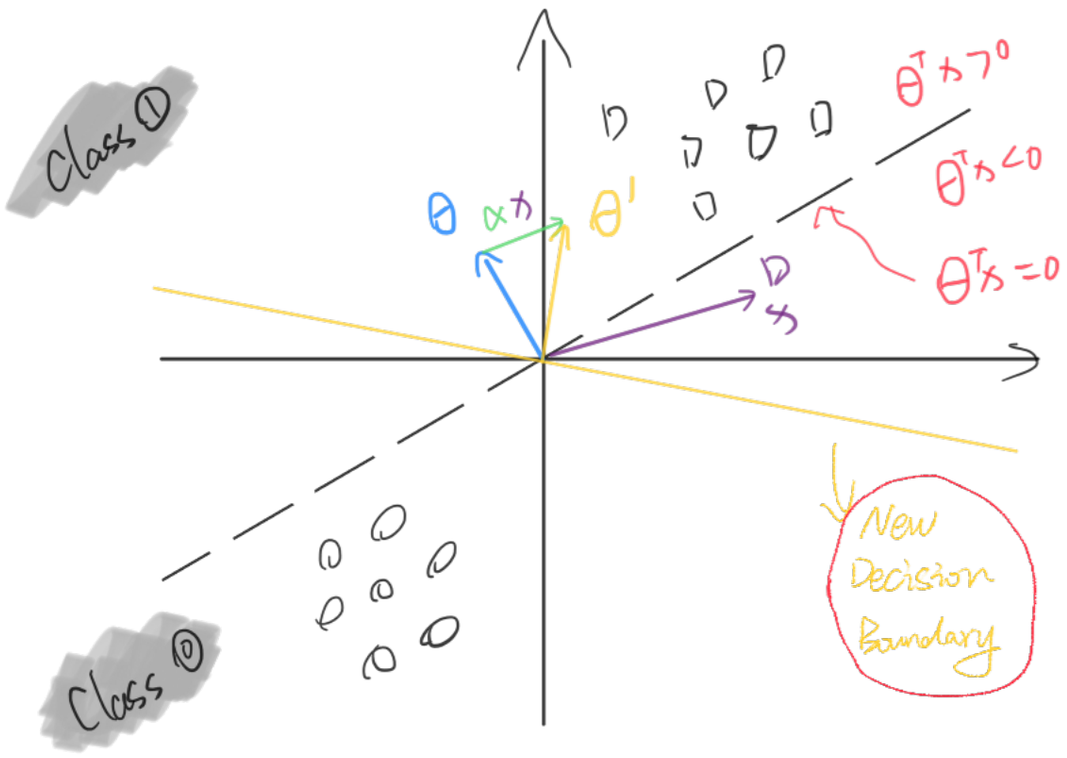
UPDATE RULE

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta \left(x^{(i)} \right) \right) x_j^{(i)}$$

$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$

Wanted:
$\theta \approx x \mid y=1$
$\theta \not\approx x \mid y=0$

$= \begin{cases} 0 & \text{if algo is right} \\ +1 & \text{if wrong \& } y^{(i)}=1 \\ -1 & \text{if wrong \& } y^{(i)}=0 \end{cases}$



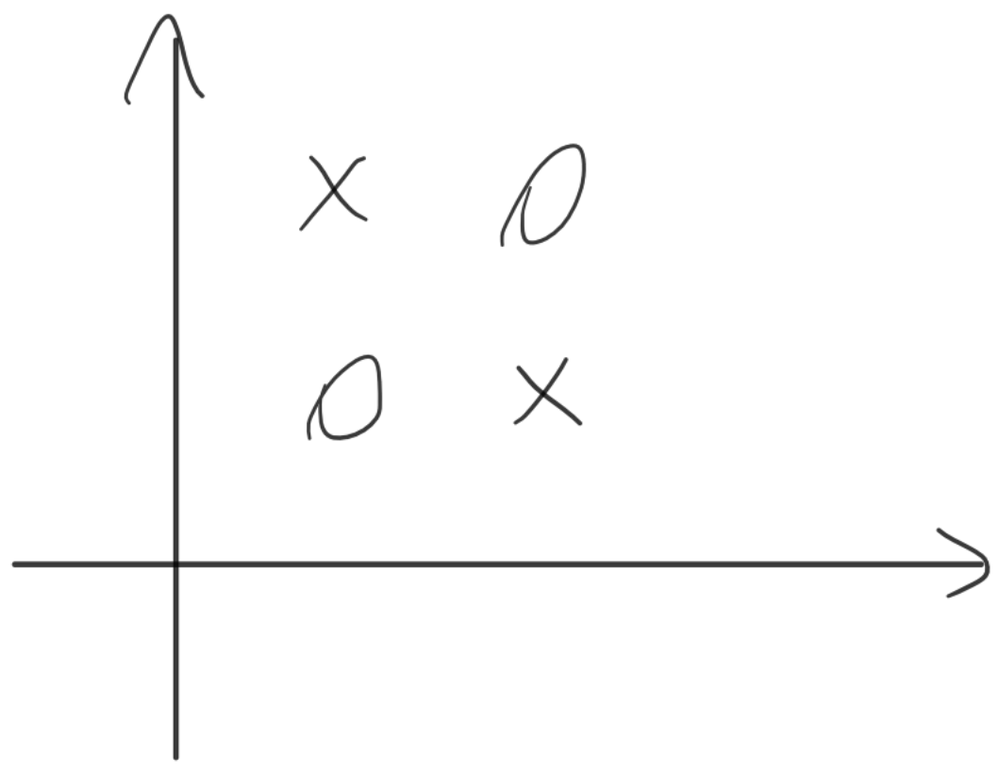
D — Newly added example, which is misclassified

When D is added, $y^{(i)} - h_\theta(x^{(i)}) = 1$

$(1 - 0 = 1)$

#Tips
If you want a vector ***A*** more similar to vector ***B*** , *you can simply add* ***B*** to ***A***

Cons:
- Cannot be used in practice, because perceptron doesn't't have probabilistic interpretation of what's happening, just have a geometrical feel
- Cannot deal with the following situation

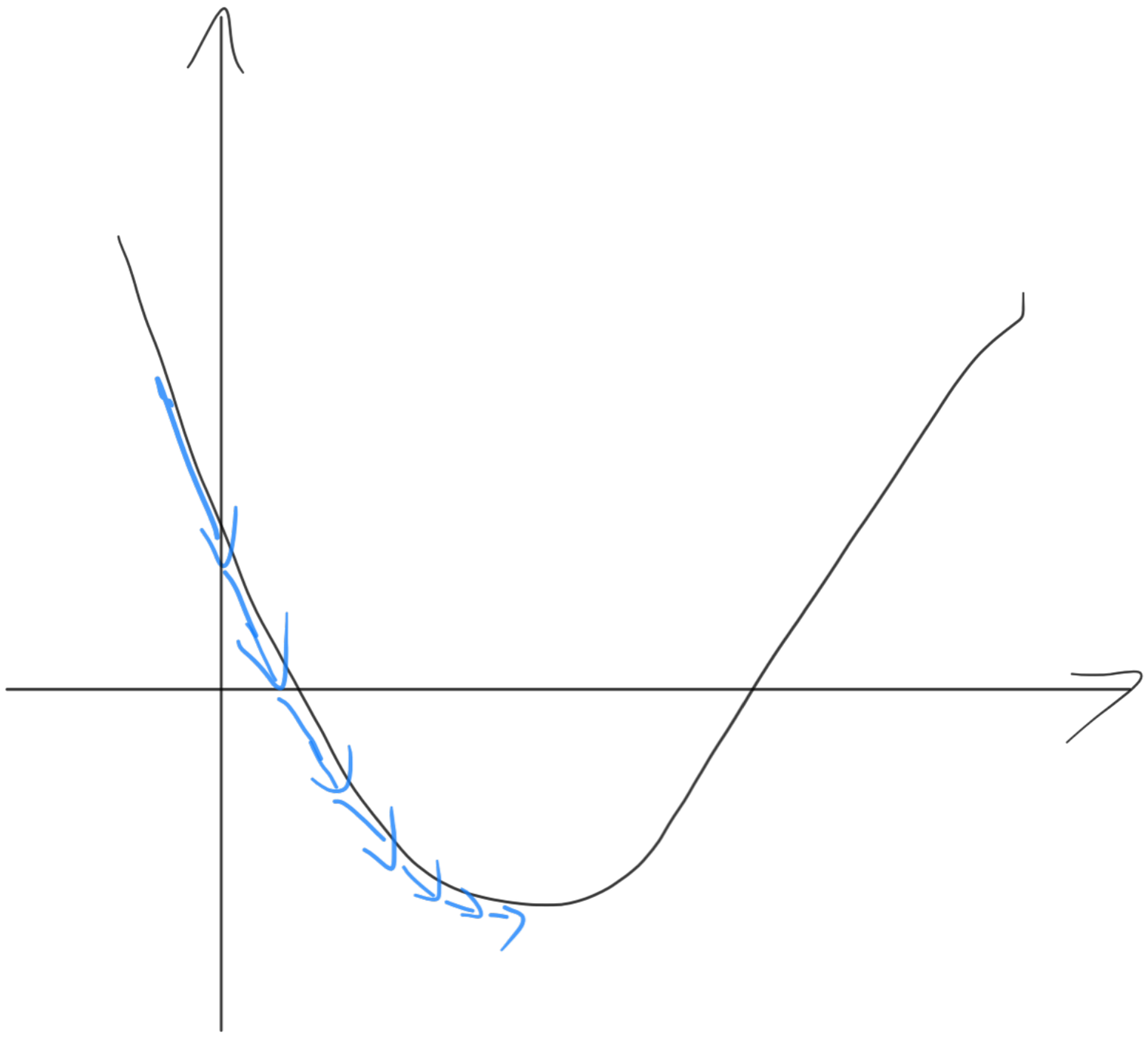


LMS algorithm

4

LMS stands for "least mean squares"

gradient descent



- batch gradient descent

$$\text{Repeat until convergence } \{$$
$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{).}$$
$$\}$$

looks at every example in the entire training set on every step

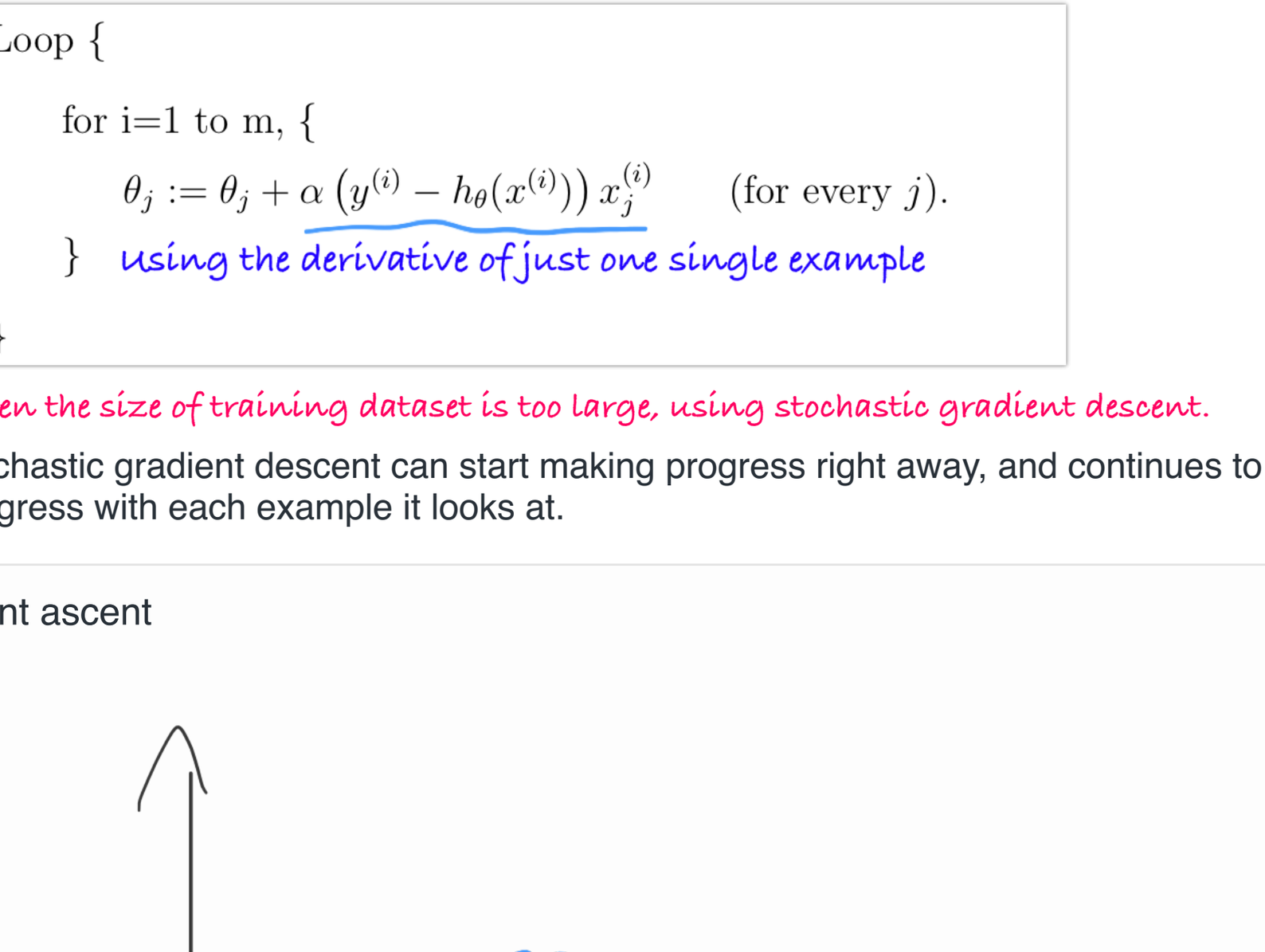
- stochastic gradient descent

$$\text{Loop } \{$$
$$\text{for i=1 to m, } \{$$
$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{).}$$
$$\} \quad \text{using the derivative of just one single example}$$
$$\}$$

When the size of training dataset is too large, using stochastic gradient descent.

stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at.

gradient ascent



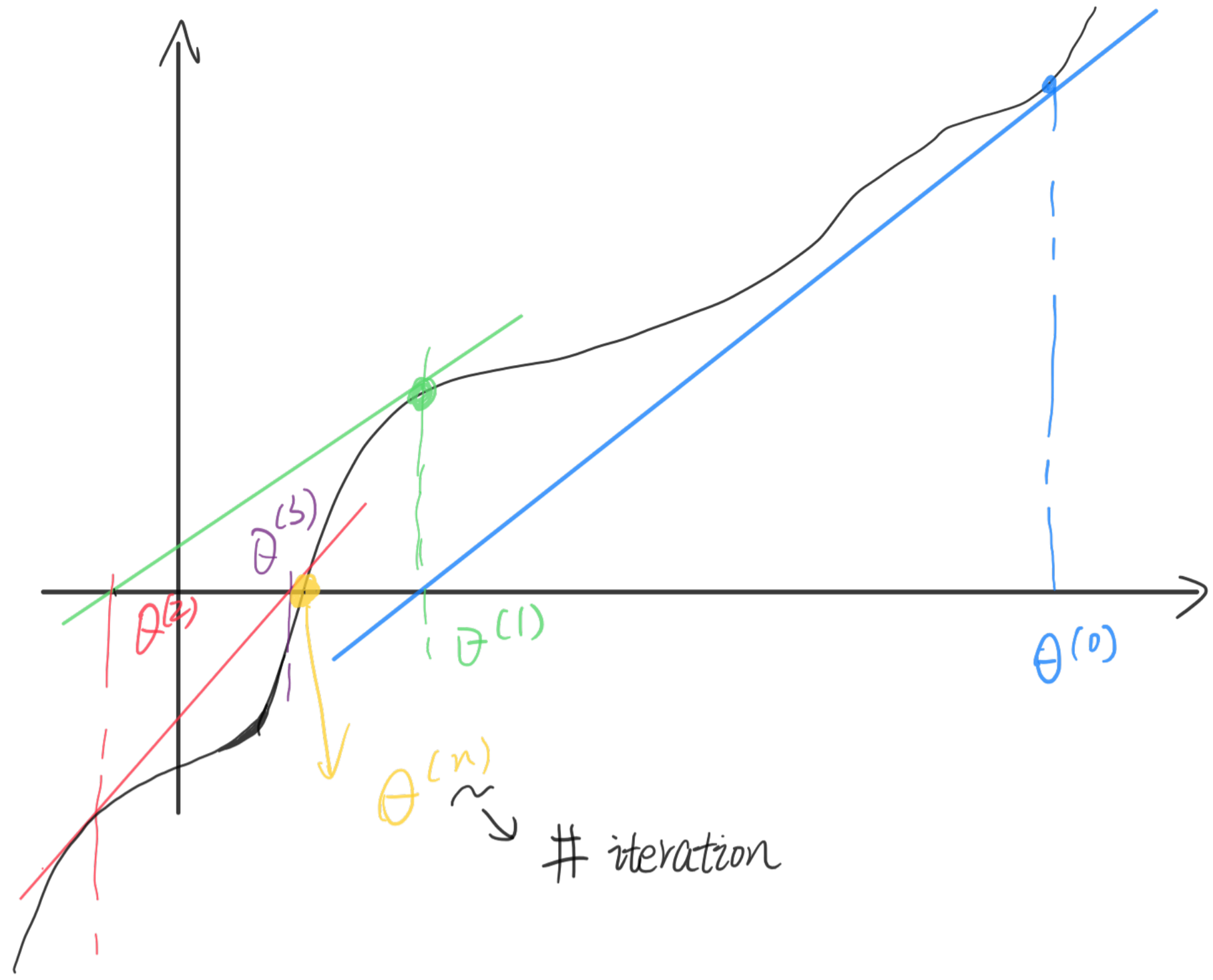
- stochastic gradient ascent

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta \left(x^{(i)} \right) \right) x_j^{(i)}$$

- gradient ascent

$$\theta_j := \theta_j + \alpha \sum_{j=1}^{m} \left(y^{(i)} - h_\theta \left(x^{(i)} \right) \right) x_j^{(i)}$$

- Newton's method



When Θ is a real number

$$For\ 1st\ \theta, \theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

$$For\ its\ predecessor, \theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$

When Θ is a vector

$$\theta := \theta - H^{-1} \nabla_\theta \ell(\theta)$$

$$H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}$$

Newton's Method has quadratic convergence property.

PROS:
Newton's method needs relatively fewer iteration steps (compared with Gradient ascent) to converge.
CONS:
In high-dimensional problems, if θ is a vector, each step of Newton's Method oils much more expensive.

Generalized Linear Models

No matter what kind of GLM/which choice of distribution that you make, **the learning update rule is the same, just need different** $h_\theta(x^{(i)})$

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta \left(x^{(i)} \right) \right) x_j^{(i)}$$

Assumptions / Design Choices

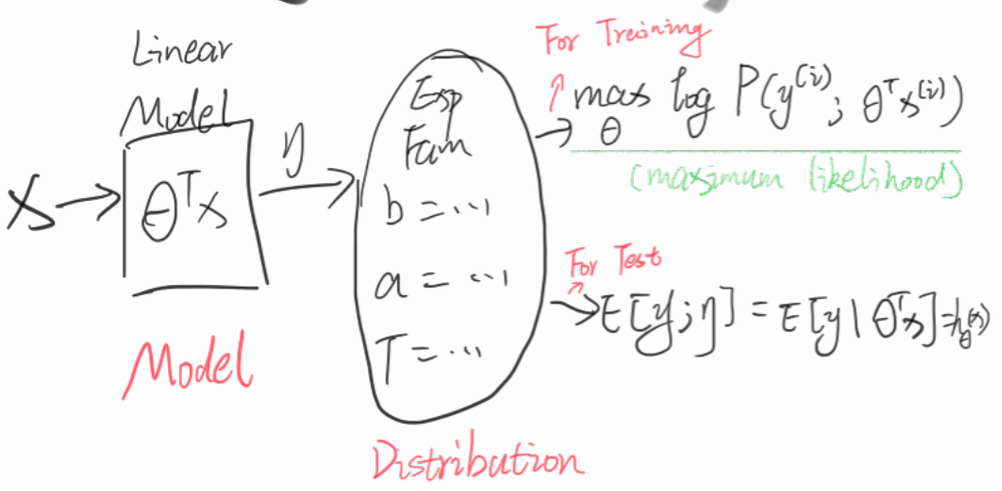
1) $y|x;\theta \sim$ Exponential Family (η)

2) $\eta = \theta^T x$ $\quad \theta \in \mathbb{R}^n$ $\quad x \in \mathbb{R}^n$

3) Test time : output $E[y|x;\theta]$

$\Rightarrow h_\theta(x) = E[y|x;\theta]$

The Big Picture for GLM



b, a, T based on the distribution of your choice (Classification, regression...)

Training Θ to predict the parameter of the exponential family distribution whose mean is the prediction that we gonna make for y

Terminology

natural parameter — η

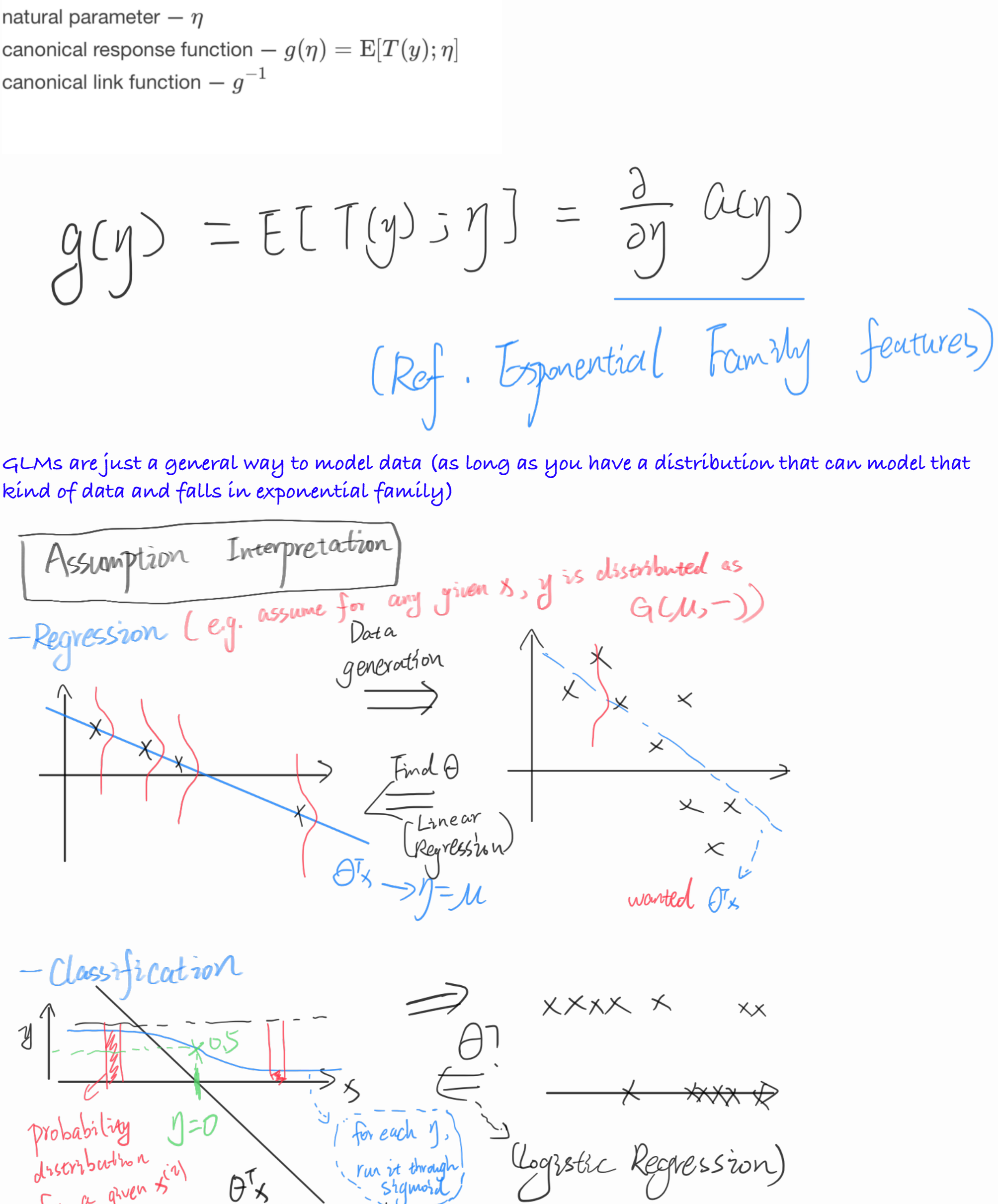
canonical response function — $g(\eta) = \mathrm{E}[T(y);\eta]$

canonical link function — g^{-1}

$$g(\eta) = E[T(y);\eta] = \frac{\partial}{\partial \eta} a(\eta)$$

(Ref. Exponential Family features)

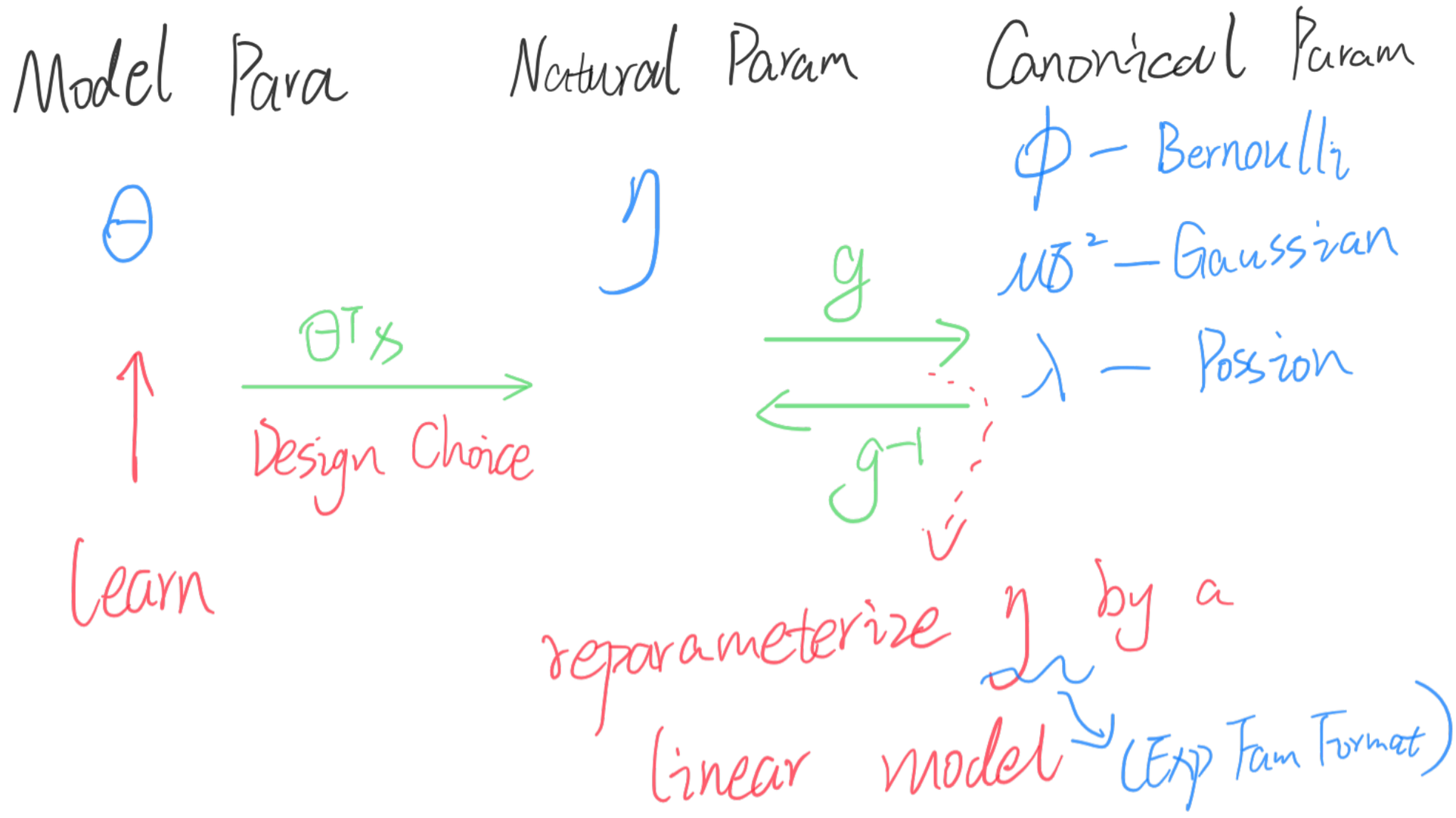
GLMs are just a general way to model data (as long as you have a distribution that can model that kind of data and falls in exponential family)



- 3 Different Ways to Parameterize GLM

  Generalized Linear Models

  exponential family



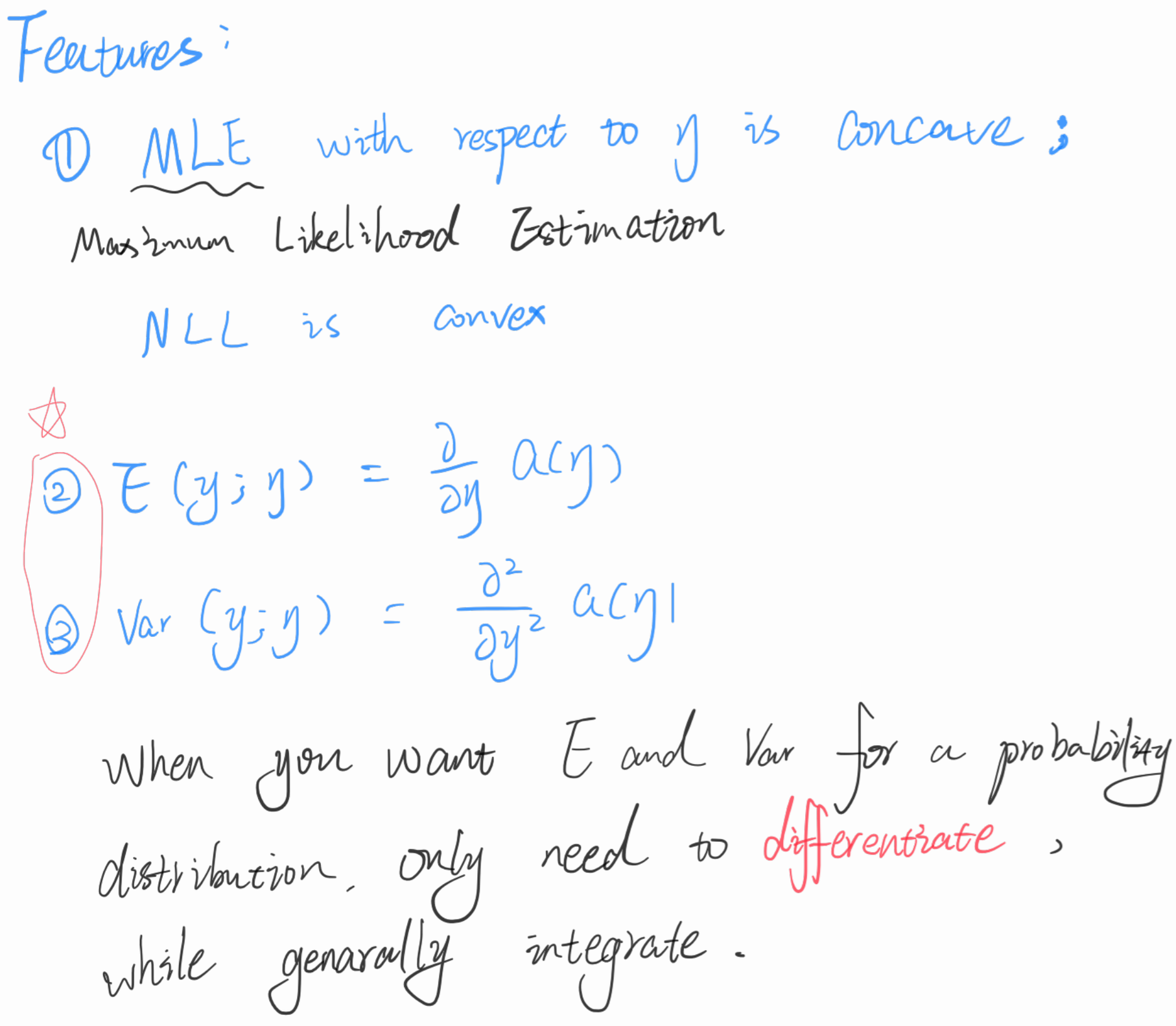
E.g. When we do logistic regression, we have

$$h_\theta(x) = E[y|x;\theta] = \phi == \frac{1}{1+e^{-\theta^T x}} = \frac{1}{1+e^{-\eta}}$$

- exponential family

  a class of distributions is in the exponential family if it can be written in the form
  (e.g. Bernoulli distribution is a member of the exponential family)

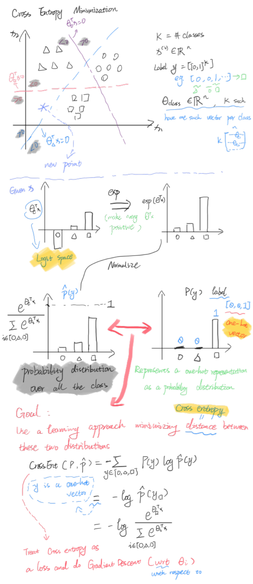
$$p(y;\eta) = b(y)\exp\left(\eta^T T(y) - a(\eta)\right)$$



Some common exponential distribution:

Real — Gaussian
Binary — Bernoulli
Count — Poisson
R^+ — Gamma, Exponential
Distribution — beta, Dirichlet [Bayesian statistics]

- Softmax Regression



Generative Learning algorithms

Gaussian discriminant analysis (GDA)

classification problem in which the input features x are continuous-valued random variables

GDA is computationally efficient

Modeling

The GDA model is :

$$y \sim \text{Bernoulli}(\phi)$$
$$x \mid y = 0 \sim \mathcal{N}(\mu_0, \Sigma)$$
$$x \mid y = 1 \sim \mathcal{N}(\mu_1, \Sigma)$$

The log-likelihood is :

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \log \prod_{i=1}^{m} p\left(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma\right)$$
$$= \log \prod_{i=1}^{m} p\left(x^{(i)} \mid y^{(i)}; \mu_0, \mu_1, \Sigma\right) p\left(y^{(i)}; \phi\right)$$

Maximum Likelihood Estimation :

$$\phi = \frac{1}{m} \sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}$$
$$\mu_0 = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\} x^{(i)}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\}}$$
$$\mu_1 = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\} x^{(i)}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}}$$
$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)} - \mu_{y^{(i)}}\right)\left(x^{(i)} - \mu_{y^{(i)}}\right)^T$$

Prediction

Goal : output what you think of y for an given x, ***rather than put put a probability***.

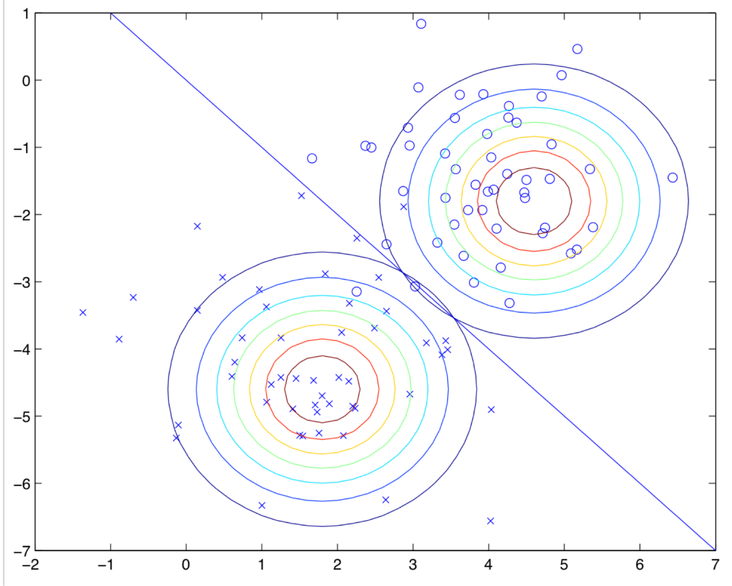
$$\arg\max_y p(y \mid x) = \arg\max_y \frac{p(x \mid y)p(y)}{p(x)}$$
$$= \arg\max_y p(x \mid y)p(y)$$

* FYI

arg max/min is the value you need plug in to achieve (i.e. argument) that max/min value.

e.g. $\min (z-5)^2 = 0$

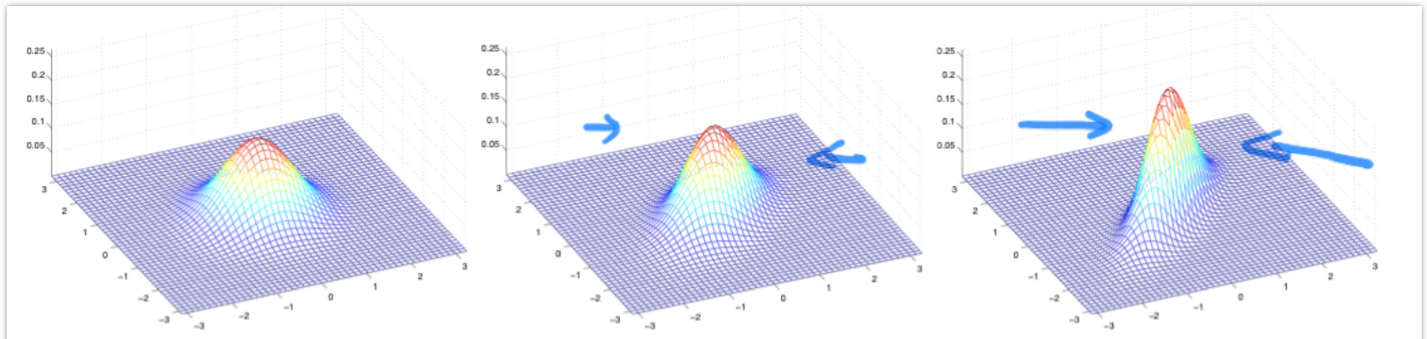
$\arg\min (z-5)^2 = 5$



use same covariance matrix Σ (for y=0 and y=1), the decision end up being linear, different Σ is also okay to build up decision boundary, except it won't be linear anymore.

- multivariate normal distribution / multivariate Gaussian distribution

33



The figures above show Gaussians with mean 0, and with covariance matrices respectively

x_1 & x_2 are positively correlated

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}.$$

Naive Bayes

38

Very efficient (slower than Logistic Regression though) and also keep on updating even as you get new data.

SCENARIO: face a problem and implement something quick and dirty to solve it

Naive Bayes (NB) assumption

assume that the xi's are conditionally independent given y.

Joint Likelihood

$$\mathcal{L}\left(\phi_y, \phi_{j|y=0}, \phi_{j|y=1}\right) = \prod_{i=1}^{m} p\left(x^{(i)}, y^{(i)}\right)$$

MLE

$$\phi_{j|y=1} = \frac{\sum_{i=1}^{m} 1\left\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\right\}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}}$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^{m} 1\left\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\right\}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\}}$$

$$\phi_y = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}}{m}$$

PROS:
- Computationally efficient
- Relatively quick to implement
- Doesn't require iterative

CONS:
- Logistic regression work better in terms of delivering and higher accuracy than Naive Bayes

- Laplace Smoothing

41

$$\phi_j = \frac{\sum_{i=1}^{m} 1\left\{z^{(i)} = j\right\} + 1}{m + k}$$

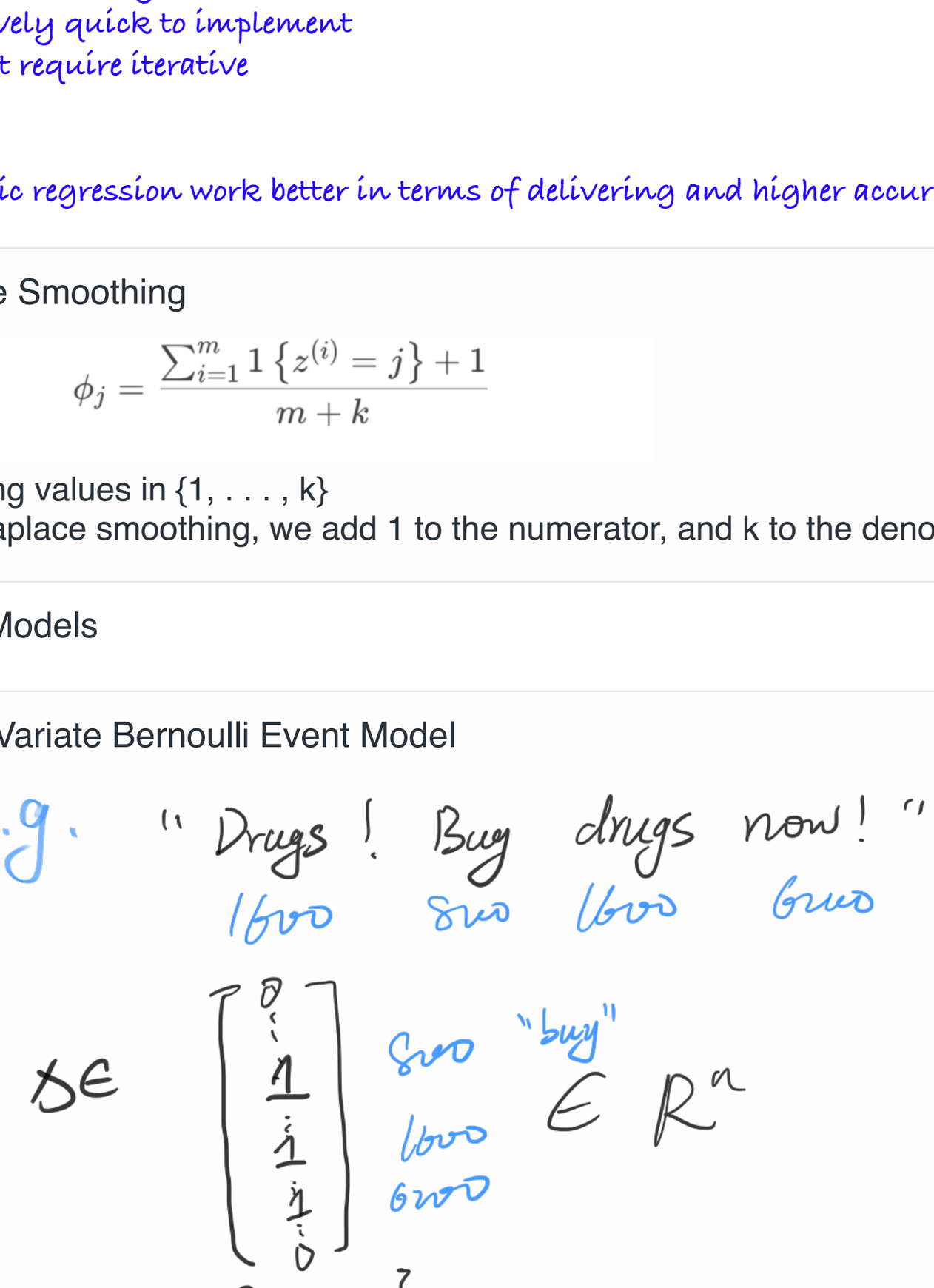
z taking values in {1, . . . , k}
For Laplace smoothing, we add 1 to the numerator, and k to the denominator

Event Models

43

- Multi-Variate Bernoulli Event Model

43



- Multinomial Event Model

43

e.g. "Drugs! Buy drugs now!"
1600 800 1600 6000

$$x \in \begin{bmatrix} 1600 \\ 800 \\ 1600 \\ 6000 \end{bmatrix} \in R^n$$

$x_j \in \{1, 2, \cdots 10000\}$

n_i = length of email

$$P(x, y) = P(x|y)P(y)$$

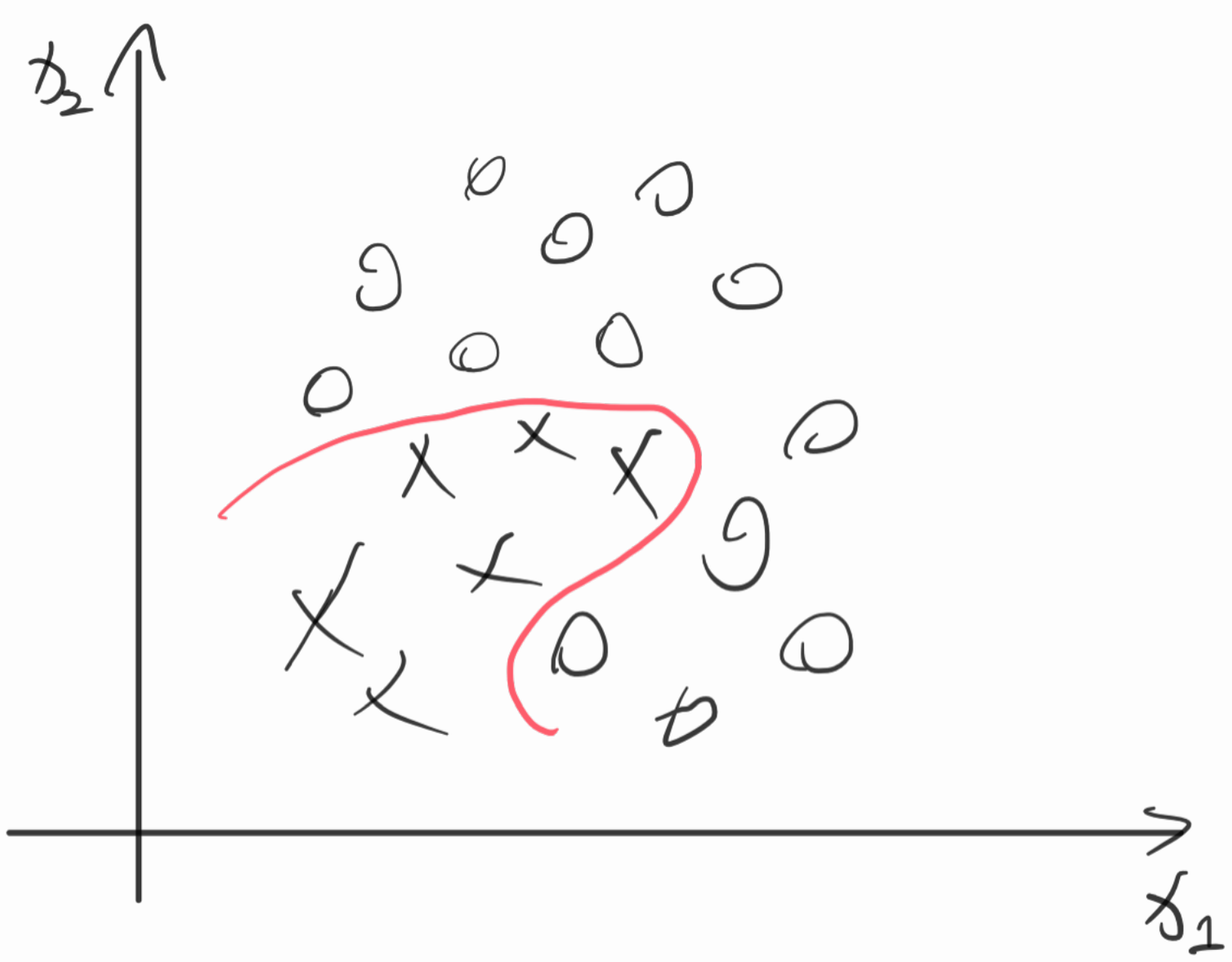
Naive Bayes assumption $\prod_{j=1}^{n} P(x_j|y_i)P(y_j)$

Support Vector Machines

45

SVM = Optimal Margin Classifier + Kernel tricks

Find a hyperplane in a high-dimension space and project that decision hyperplane into a (non-linear) decision boundary in your low-dimension space.



PROS
- Turn-key, (i.e. doesn't have many parameters to fiddle with)

CONS
- not as effective as neural network

Notation

$$y \in \{-1, 1\}$$
$$h_{w,b}(x) = g\left(w^T x + b\right)$$
$$g(z) = 1 \text{ if } z \geq 0$$
$$g(z) = -1 \text{ otherwise}$$

Due to $g(z)$, classifier will directly predict either -1 or 1.

Optimal Margin Classifiers

54

Basic building block for the supportive vector machine.

Optimal Margin Classifier is the algorithm tries to maximize the geometric margin.

$$\min_{\gamma,w,b} \frac{1}{2}\|w\|^2$$
$$\text{s.t. } y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1, \quad i = 1, \ldots, m$$

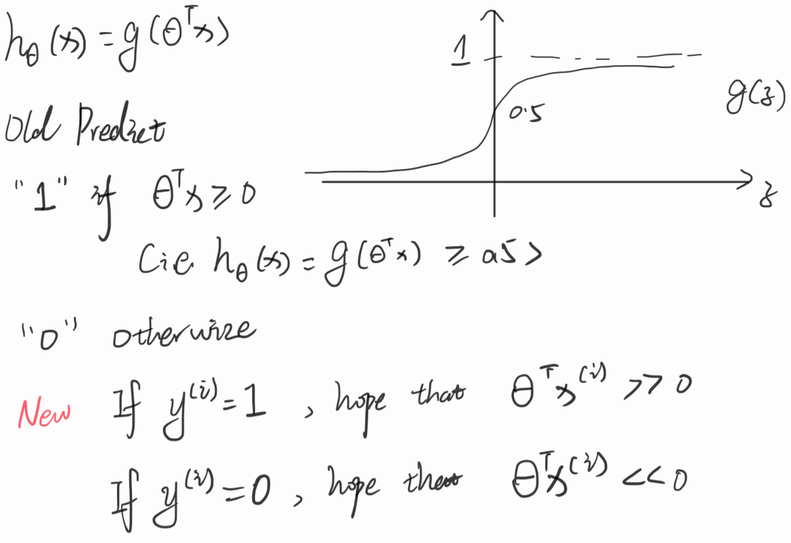
Dual Optimization Problem

$$max_\alpha\, W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \left\langle x^{(i)}, x^{(j)} \right\rangle$$
$$\text{s.t. } \alpha_i \geq 0, \quad i = 1, \ldots, m$$
$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

- Functional Margin

47

How confidently and accurately do you classify an example



Functional Margin

$$\hat{\gamma}^{(i)} = y^{(i)}\left(w^T x + b\right)$$
$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)}$$

Attention

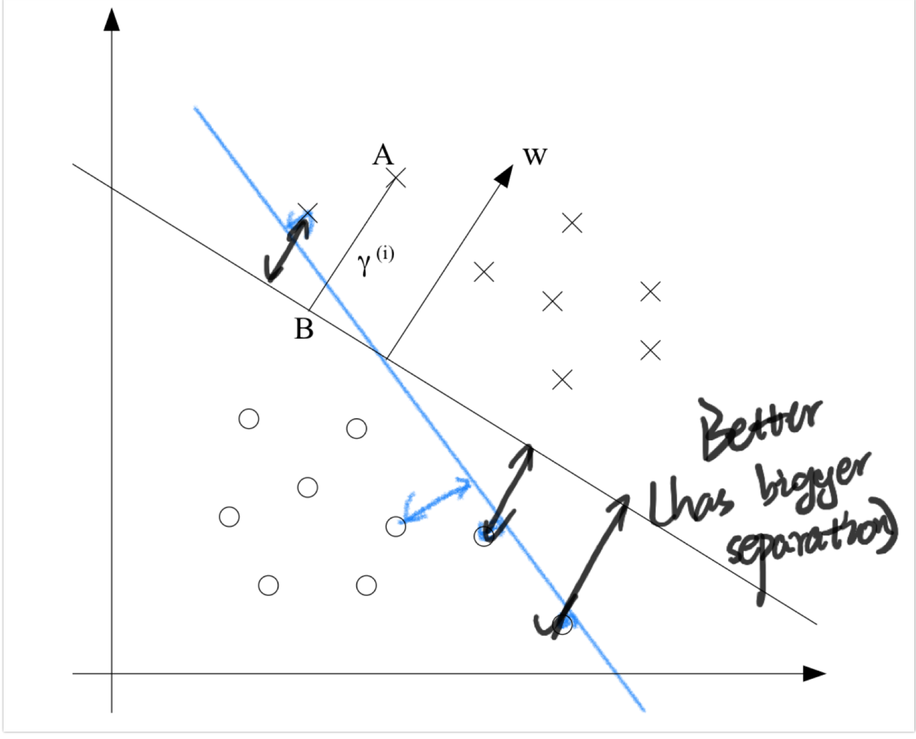
$h_{w,b}(x) = g\left(w^T x + b\right)$ depend only on the signal but not magnitude of $w^T + b$– **Use normalization to solve this situation**

Normalization

$$\text{replace } (w, b) \text{ with } (w/\|w\|_2, b/\|w\|_2)$$

- Geometric Margin

48



Geometric Margin

$$\gamma^{(i)} = y^{(i)}\left(\left(\frac{w}{\|w\|}\right)^T x^{(i)} + \frac{b}{\|w\|}\right)$$
$$\gamma = \min_{i=1,\ldots,m} \gamma^{(i)}$$

if $\|w\| = 1$, geometric margin is equal to functional margin.

- Geometric Margin is invariant to rescaling parameters.

- Kernels

57

Map low-dimensional feature vector into high-dimensional (maybe infinite) set of feature vector.

PROS

- Relieves us from manually picking features

Kernel Trick:

1. Write algorithm in terms of $< x^{(i)}, x^{(j)} >$ (or $< x, z >$)
2. Let there be some mapping from your original input features $x \rightarrow \phi(x)$
3. Find way to compute $K(x, z) = \phi(x)^T \phi(z)$ – **Kernel Function**
4. Replace $< x, z >$ in algorithm with $K(x, z)$

   If you doing this replace, you 're running the whole learning algorithm on high dimensional set of features

Common Kernel Functions

Gaussian Kernel *(Mostly used)*

$$K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$$

Polynoimal Kernel

$$K(x,z) = (x^T z)^d$$

- Non-Separable Case

63

- L1 Regularization

63

> l1 regularization

$$min_{\gamma,w,b}\ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i$$

$$\text{s.t. } y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \xi_i, \quad i = 1, \ldots, r$$

$$\xi_i \geq 0, \quad i = 1, \ldots, m.$$

Dual form for L_1 Norm Soft Margin SVM

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j \left\langle x^{(i)}, x^{(j}\right.$$

$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m$$

$$\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$$

make the algorithm work for non-linearly separable datasets as well as be less sensitive to outliers

- L2 Regularization

173

- Empirical Risk Minimization (ERM)

73

$$\hat{\varepsilon}(h) = \frac{1}{m}\sum_{i=1}^{m} 1\left\{h\left(x^{(i)}\right) \neq y^{(i)}\right\}$$

$$\hat{\theta} = \arg\min_{\theta} \hat{\varepsilon}\left(h_\theta\right)$$

- Hoeffding Inequality

$$P\left(\left|\varepsilon\left(h_i\right)-\hat{\varepsilon}\left(h_i\right)\right|>\gamma\right) \leq 2 \exp\left(-2 \gamma^2 m\right)$$

Decision Trees

- Cross Entropy Loss

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{k} y_j \log \hat{y}_j$$

- Ensembling

Ensembling

Take X_i's which are random variables that are independent identically distributed (IID)

$$Var(X_i) = \sigma^2$$

$$Var(\bar{X}) = Var(\frac{1}{n}\sum_i X_i) = \frac{\sigma^2}{n}$$

Drop the independence assumption, so now X_i's just identically distributed(ID), X's correlated by ρ.

$$Var(\bar{X}) = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

Ways to Ensemble

1. ~~different algorithms~~
2. ~~different training sets~~
3. Bagging (*e.g. Random Forests*)
   Try to approximate having different training sets.
4. Boosting (*e.g. AdaBoost, XGBoost*)

Bagging - Bootstrap Aggregation

Take a bunch of bootstrap samples, train separate models on each and then average their outputs.

Bootstrap

1. Have a true population P
2. Training set $S \sim P$
3. Assume $P = S$
4. Bootstrap samples $Z \sim S$

Bootstrap Aggregation

1. Bootstrap samples $Z_1, Z_2, Z_3..., Z_M$
2. Train model G_m on Z_m
3. Define a meta model $G(m) = \frac{\sum_{m=1}^{M} G_m(x)}{M}$

Bias-Variance Analysis

$Var(\bar{X}) = \rho\sigma^2 + \frac{1-\rho}{M}\sigma^2$

- Bootstrapping is driving down ρ
  Increasing the number of bootstrap models in your training, doesn't cause you to overfit anymore than you were beforehand.
- More M leads to less variance. *(There is a lower bound, can't make variance 0)*
- Bias is slightly increased because of random subsampling. *(Because the bootstrap samples Z are actually subsets of the original set S, so you model become less complex, that increases bias)*

Decision trees + Bagging

Decision trees are high variance, low bias.

Ideal fit for bagging

Random Forests

At each split, consider only a fraction of total features.

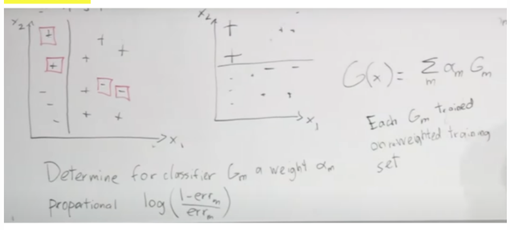
- Decrease ρ
- Decorrelate Models

Boosting

Adaboost, XGBoost, gradient boost machine

- Decrease bias
- Additive

AdaBoost



DeepLearning

- Xavier/He Initialization

170

$$w^{[\ell]} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n^{[\ell]} + n^{[\ell-1]}}}\right)$$

Unsurpervised Learning Algorithms

Expectations-Maximization

95

Solve MLE directly might be hard (Because z is a latent variable). Instead, EM repeatedly construct a lower-bound on l (E-step), and then optimize that lower-bound (M-step).

EM implements a softer way of assigning points to the different cluster centroids.

- To do EM, we need a concave function

1. **E-step**

$$Q_i(z^{(i)}) = P(z^{(i)} \mid x^{(i)}; \theta)$$

$Q_i(z^{(i)})$ *is* $w_j^{(i)}$

2. **M-step**

$$\theta := \arg\max_\theta \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)};}{Q_i(z^{(i)})}$$

Iterative E-step and M-step, the algorithm should converge to a local optima.

Question: Why don't we just take $\arg\max_\theta l(\theta)$ *?*

Because there's no known way to solve that.

Repeat until convergence: {

(E-step) For each i, j, set $w^{(i)}_j$ is how much $x^{(i)}$ is assigned to μ_j Gaussian.

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

(M-step) Update the parameters:

$$\begin{aligned}
\phi_j &:= \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)}, \\
\mu_j &:= \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}}, \\
\Sigma_j &:= \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}
\end{aligned}$$

}

$$p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{\sum_{l=1}^{k} p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)}$$

- The K-Means Clustering Algorithm

1. Initialize **cluster centroids** $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$ randomly.
2. Repeat until convergence: {

"Color the points"

For every i, set

$$c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$$

"Moves the cluster"

For each j, set

$$\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}}.$$

}

How to choose K ?

Issue: `#clusters` might be ambiguous.

Solution:

1. AIC or BIC criteria for automatically choosing `#clusters`
2. **Choose manually**

What to do when K-means stuck in local minima?

Run K-means on different iteration times and different initializations of cluster centroids. Pick the lowest cost function $J(c, \mu)$ run.

- Mixture of Gaussians

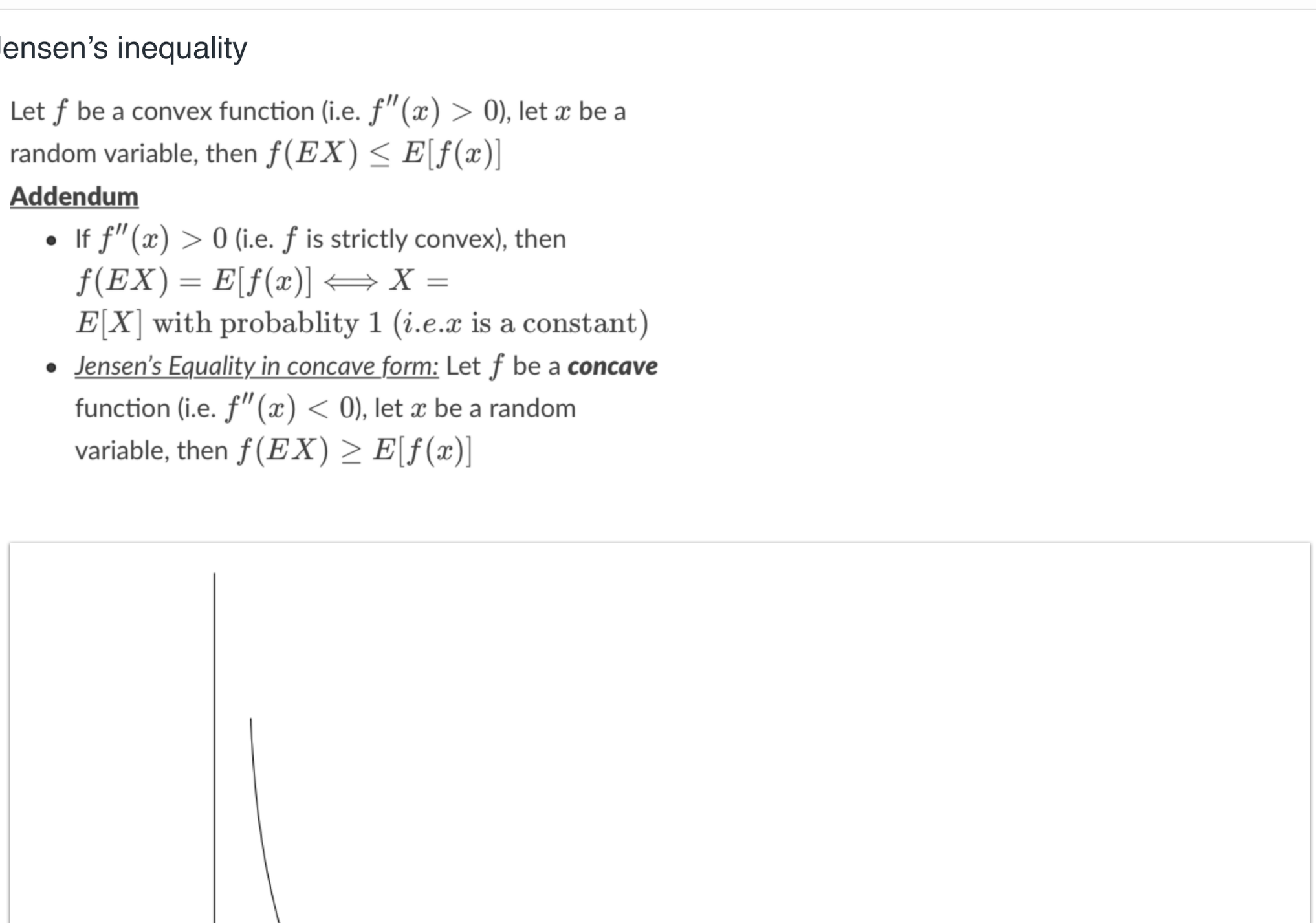
model posits that each x(i) was generated by randomly choosing z(i) from {1, . . . , k}, and then x(i) was drawn from one of k Gaussians depending on z(i).

- Jensen's inequality

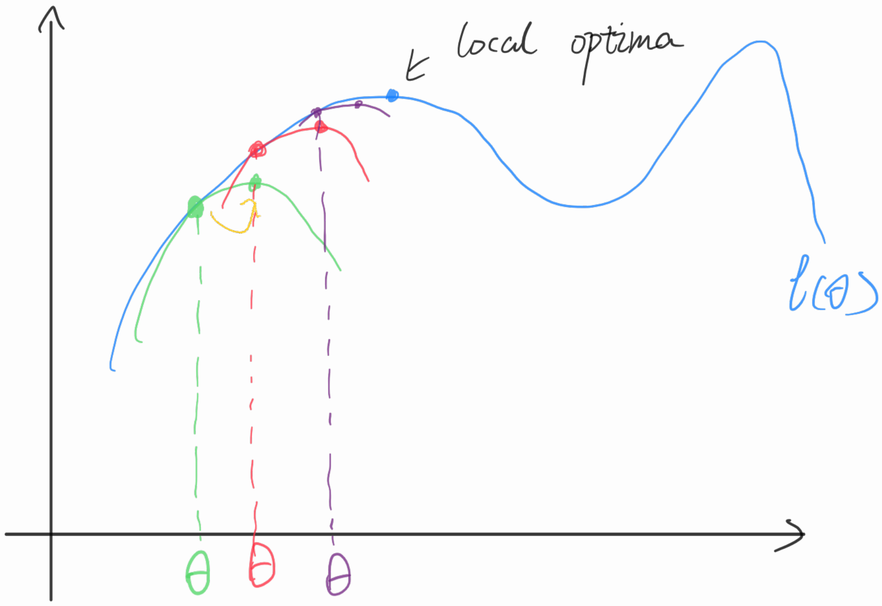
Let f be a convex function (i.e. $f''(x) > 0$), let x be a random variable, then $f(EX) \leq E[f(x)]$

Addendum

- If $f''(x) > 0$ (i.e. f is strictly convex), then $f(EX) = E[f(x)] \Longleftrightarrow X = E[X]$ with probablity 1 $(i.e. x$ is a constant$)$
- *Jensen's Equality in concave form:* Let f be a ***concave*** function (i.e. $f''(x) < 0$), let x be a random variable, then $f(EX) \geq E[f(x)]$



- Density Estimation



EM algorithm visualization:

- At E-step, constructing a lower bound (green curve) for the log-likelihood. Green curve has two properties
  1. Green curve is a lower bound (i.e. green curve lies blow the blue curve)
  2. Its value is equal to the blue curve at the current value of θ. (*This property guarantees that when you optimize the green function, you can improving blue function too.*)

     $$\log E_{z^{(i)}\sim Q_i}\left[\frac{P(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}\right] = E_{z^{(i)}\sim Q_i}\left[\log \frac{P(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}\right]$$

     To ensure this property, we need

     $$Q_i(z^{(i)}) = P(z^{(i)} \mid x^{(i)};\theta)$$
- At M-step, take the green curve and find its maximum
  Move θ from green value to the red value.

Factor Analysis

107

Factor Analysis $z^{(i)} \sim \mathcal{N}$

Factor analysis can take very high dimensional data and model them to a lower dimensional subspace with a little bit of fuzz.

If the data doesn't lie in a subspace, model may not be the best model. But it's still a reasonable way to fit a Factor Analysis to n>>m dataset.

[Time Label]

A continuous z EM model

- **One Other View of EM**
- Coordinate Ascent $J(\theta, Q)$
- Comparison between Mixture of Gaussian & Factor Analysis
  - When $m \gg n$, use MG
    img
  - else FA

If `#Training Examples` < the dimension of the data, usual MLE function of covariance Σ will be singular(non-invertible)

- Marginals and Conditionals of Gaussians

image-20220425150407136

Marginal : calculate $P(x_1)$ and we have $x_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$

Conditional : calculate $P(x_1 \mid x_2)$ we have $x_1 \mid x_2 \sim \mathcal{N}(\mu_{1|2}, \Sigma_{1|2})$

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$$

$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

- EM for Factor Analysis

EM Steps

1. Derive $P(x, z)$

$$\begin{pmatrix} z \\ x \end{pmatrix} \sim \mathcal{N}(\mu_{x,z}, \Sigma)$$
$$z \sim \mathcal{N}(0, 1)$$
$$x = \mu + \Lambda z + \epsilon$$

image-20220425153323755

Simplify above equations. Finally we have

$$\mu_{x,z} = \begin{pmatrix} 0 \\ \mu \end{pmatrix}$$
$$\Sigma = \begin{pmatrix} I & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \Psi \end{pmatrix}$$

Putting everything together, we have

$$\begin{bmatrix} z \\ x \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \vec{0} \\ \mu \end{bmatrix}, \begin{bmatrix} I & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \Psi \end{bmatrix}\right)$$

There's no parameters' closed form when solve derivatives of $P(x^{(i)})$'s log likelihood.

So, we use EM to solve these parameters.

2. **EM for Factor Analysis**
   - **E-Step**

$$Q_i(z^{(i)}) = P(z^{(i)} \mid x^{(i)}; \theta)$$
$$z^{(i)} \mid x^{(i)} \sim \mathcal{N}(\mu_{z^{(i)}|x^{(i)}}, \Sigma_{z^{(i)}|x^{(i)}})$$

Where

$$\mu_{z^{(i)}|x^{(i)}} = \vec{0} + \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}(x^{(i)} - \mu)$$
$$\Sigma_{z^{(i)}|x^{(i)}} = I - \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}\Lambda$$

   - **M-step**

$$\Lambda = \left(\sum_{i=1}^{m}\left(x^{(i)} - \mu\right)\mu_{z^{(i)}|x^{(i)}}^T\right)\left(\sum_{i=1}^{m}\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}}\right)^{-1}$$
$$\mu = \frac{1}{m}\sum_{i=1}^{m}x^{(i)}$$
$$\Psi = \frac{1}{m}\sum_{i=1}^{m}x^{(i)}x^{(i)^T} - x^{(i)}\mu_{z^{(i)}|x^{(i)}}^T\Lambda^T - \Lambda\mu_{z^{(i)}|x^{(i)}}x^{(i)^T} + \Lambda\left(\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}}\right)$$

Tips:

Simplify integrals by

$$\int_{z^{(i)}} Q_i(z^{(i)})z^{(i)}dz^{(i)} = E[z^{(i)}] = \mu_{z^{(i)}|x^{(i)}}$$

Focus on finding V^*
For absorbing state, set its P_{sa} to 0
Value iteration converges very quickly. (Due to γ, converges exponentially quickly)

> 1. For each state s, initialize $V(s) := 0$.
> 2. Repeat until convergence {
>
> For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s')V(s')$
>
> }

repeatedly trying to update the estimated value function using Bellman Equations

first compute the new values for V (s) for every state s, and then overwrite all the old values with the new values

loop over the states (in some order), updating the values one at a time

Focus on finding π^*

> 1. Initialize π randomly.
> 2. Repeat until convergence {
>
> (a) Let $V := V^{\pi}$.
> (b) For each state s, let $\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s')V(s')$.
>
> }

inner-loop repeatedly computes the value function for the current policy, and then updates the policy using the current value function

- (Direct) Policy Search

Solve π^* directly (instead of using V^* to solve π^*)
DPS focus on coming up with the class of policies you'll entertain or come up with the set of functions you use to approximate the policy.
New definition: A stochastic policy is a function $\pi : S \times A \longmapsto \mathbb{R}$ when $\pi(s, a)$ is the probability of taking action a in state s ($\sum_a \pi(s, a) = 1$)
Goal : Find θ so that when we execute $\pi\theta(s, a)$, we maximize total payoff (expected sum of rewards), i.e.

$$\max_{\theta} E\left[R\left(s_0, a_0\right) + \ldots + R\left(s_T, a_T\right) \mid \pi\theta\right]$$

How to do that Derive a stochastic gradient ascent algorithm as a function of θ solve the equation above.
Reinforce Algorithm (*Very inefficient*)

```
loop{
/*Sample*/
s_0, a_0, s_1,a_1 ... s_T,a_T
/* Compute payoff*/
$R(s_0,a_0),...,R(s_T,a_T)$
/*Update $\theta$ (using gradient ascent)*/
Implement $\theta$ update rule
}
```

θ update rule

$$\theta := \theta + \alpha \left[\frac{\nabla \pi\theta(s_0, a_0)}{\pi\theta(s_0, a_0)} + \frac{\nabla \pi\theta(s_1, a_1)}{\pi\theta(s_1, a_1)} + \ldots + \frac{\nabla \pi\theta(s_T, a_T)}{\pi\theta(s_T, a_T)}\right] (R(s_0, a_0), \ldots, R(s_T, a_T)$$

Note: One difference between policy search and estimated value function is that in Direct policy search s_0 is a fixed initial state s_0 or there's a fixed distribution over initial state s_0.
Direct policy search also works for continuous value function, in that situation, we have $a = \theta^T +$ Gaussian noise.

- Direct Policy Search vs Value-function-based Approach

  (Direct) Policy Search

  Value Function Approximation

  When to use DPS:

  1. **POMDP (Partially Observable MDP)**
     At each step, get a partial (and potentially noisy) measurement of the state. Have to choose an action a using that.
     If we just have partially observed value of the state, even if we know $V^\star(s), \pi^\star(s)$, because we can't ensure what the state is, we still can not apply then. So we can only use DPS
     Can apply Kalman filter to estimate full state vector using partial observation state vectors, and plug them as features into policy search.
  2. **When $\pi^\star$ is simpler than $V^\star$**
     For low-level control task, we often have simple map form $\mathcal{S} \longmapsto \mathcal{A}$ i.e. simpler policy $\pi^\star$. For multi-step reasoning problems, we should prefer to choose value function based approaches.

  CONS for Reinforce Algorithm

  - **Very inefficient**
    Gradient estimate for reinforcement algorithm turns out to be very noisy, even though the expected value is right.

Continuous State MDPs 134

- Discretization 135

  discretization usually works extremely well for 1d and 2d problems (and has the advantage of being simple and quick to implement).

Value Function Approximation 137

- Fitted Value Iteration 140

1. Randomly sample m states $s^{(1)}, s^{(2)}, \ldots s^{(m)} \in S$.
2. Initialize $\theta := 0$.
3. Repeat {

   For $i = 1, \ldots, m$ {

   For each action $a \in A$ {

   Sample $s'_1, \ldots, s'_k \sim P_{s^{(i)}a}$ (using a model of the MDP).

   Set $q(a) = \frac{1}{k} \sum_{j=1}^{k} R(s^{(i)}) + \gamma V(s'_j)$

   // Hence, $q(a)$ is an estimate of $R(s^{(i)}) + \gamma \mathrm{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$.

   }

   Set $y^{(i)} = \max_a q(a)$.

   // Hence, $y^{(i)}$ is an estimate of $R(s^{(i)}) + \gamma \max_a \mathrm{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$.

   }

   // In the original value iteration algorithm (over discrete states)

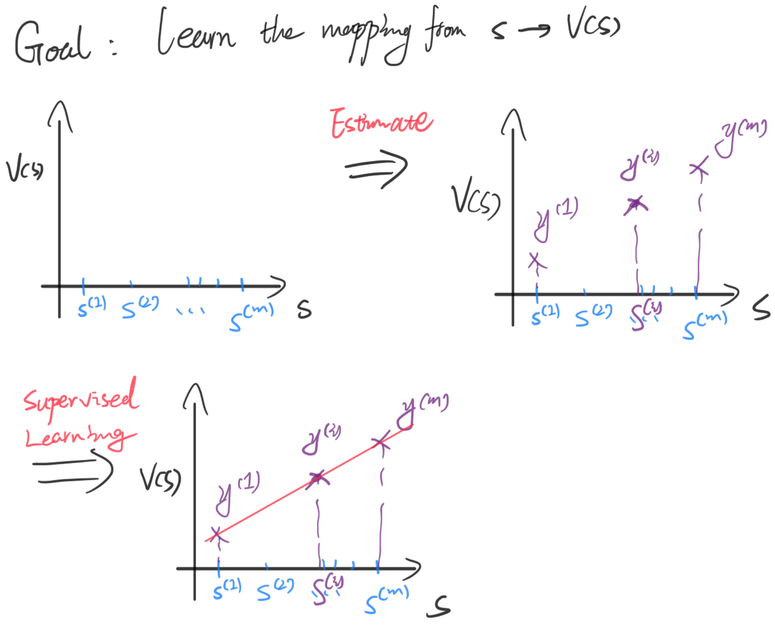
   // we updated the value function according to $V(s^{(i)}) := y^{(i)}$.

   // In this algorithm, we want $V(s^{(i)}) \approx y^{(i)}$, which we'll achieve

   // using supervised learning (linear regression).

   Set $\theta := \arg\min_\theta \frac{1}{2} \sum_{i=1}^{m} \left(\theta^T \phi(s^{(i)}) - y^{(i)}\right)^2$

   }



- (Direct) Policy Search

Solve π^* directly (instead of using V^* to solve π^*)
DPS focus on coming up with the class of policies you'll entertain or come up with the set of functions you use to approximate the policy.
New definition: A stochastic policy is a function $\pi : S \times A \longmapsto \mathbb{R}$ when $\pi(s, a)$ is the probability of taking action a in state s ($\sum_a \pi(s, a) = 1$)
Goal : Find θ so that when we execute $\pi\theta(s, a)$, we maximize total payoff (expected sum of rewards), i.e.

$$\max_{\theta} E\left[R\left(s_0, a_0\right) + ... + R\left(s_T, a_T\right) \mid \pi_\theta\right]$$

How to do that Derive a stochastic gradient ascent algorithm as a function of θ solve the equation above.
Reinforce Algorithm (*Very inefficient*)

```
loop{
/*Sample*/
s_0, a_0, s_1,a_1 ... s_T,a_T
/* Compute payoff*/
$R(s_0,a_0),...,R(s_T,a_T)$
/*Update $\theta$ (using gradient ascent)*/
Implement $\theta$ update rule
}
```

θ update rule

$$\theta := \theta + \alpha \left[\frac{\nabla \pi\theta(s_0, a_0)}{\pi\theta(s_0, a_0)} + \frac{\nabla \pi\theta(s_1, a_1)}{\pi\theta(s_1, a_1)} + ... + \frac{\nabla \pi\theta(s_T, a_T)}{\pi\theta(s_T, a_T)}\right] (R(s_0, a_0), ..., R(s_T, a_T)$$

Note: One difference between policy search and estimated value function is that in Direct policy search s_0 is a fixed initial state s_0 or there's a fixed distribution over initial state s_0.
Direct policy search also works for continuous value function, in that situation, we have $a = \theta^T +$ Gaussian noise.

MDP with unknown state transition probabilities

134

1. Initialize π randomly.
2. Repeat {
   (a) Execute π in the MDP for some number of trials.
   (b) Using the accumulated experience in the MDP, update our estimates for P_{sa} (and R, if applicable).
   (c) Apply value iteration with the estimated state transition probabilities and rewards to get a new estimated value function V.
   (d) Update π to be the greedy policy with respect to V.
   }

- Exploration vs Exploitation problem

When you acting a MDP, how aggressively of how greedy should you be at just taking actions to maximize your rewards?
Using **epsilon greedy** to solve this problem.

```
1.Initialize π randomly
2.Repeat{
        (a)Execute EPSILON-GREEDY π in the MDP
        (b)Using the accumulated experience in
        mates for $P_{sa}$ (and R,if applicable
        (c)Apply value iteration with the estim
        ities and rewards to get a new estimate
        (d)Update π to be the greedy policy wit
        }
```

Questions

- Is ϵ in **epsilon greedy** have to be constant?
  No, it doesn't have to be. *Boltzmann exploration
- Can you get a reward for reaching states you've never seen before?
  Intrinsic reinforcement learning / Intrinsic Motivation.
- How many actions should you take before updating π?
  Run as frequently as it can.

- State-Action Rewards

Rewards is a function mapping from states and actions to the rewards (i.e. $R(s,a) : S \times A \longmapsto \mathbb{R}$)

In this case, Bellman's equation is

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P_{sa}(s') V^*(s') \right)$$

Can use value iteration to solve $V^*(s)$, then you can get optimal policy, which is

$$\pi^*(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s'} P_{sa}(s') V^*(s') \right)$$

- Finite Horizon MDP

$\left(\mathcal{S}, \mathcal{A}, P_{sa}^{(t)}, T, R^{(t)}\right)$

Replace discount factor γ with a horizon time T, MDP will run a finite number of T steps.

- Action you take might depend on what time it is on the clock.
  Thus π should be time dependent (i.e. $\pi_t^*(s)$) Non-stationary policy
  Non-stationary state transitions $s_{t+1} \sim P_{s_t a_t}^{(t)}$
  Non-stationary Reward $R^{(t)}(s,a)$
  Examples
  - Changing dynamics
  - Weather forecasts
  - Industrial automation

How to solve for a finite horizon MDP

1. Define the optimal value function
   $$V_t(s) = \mathbb{E}\left[R^{(t)}(s_t, a_t) + \cdots + R^{(T)}(s_T, a_T) \mid s_t = s, \pi\right]$$
   $V_t(s)$ it the total payoff start on state s at time t execute π
2. Value iteration (a dynamic programming problem in this case)
   $\forall t < T, s \in \mathcal{S}: \quad V_t^*(s) := \max_{a \in \mathcal{A}} \left[R^{(t)}(s,a) + \mathbb{E}_{s' \sim P_{sa}^{(t)}}\left[V_{t+1}^*(s')\right]\right]$

   Base case (the final step)
   $$V_T^*(s) = \max_a R(s,a)$$

   Get the optimal policy
   $$\pi_t^*(s) = \arg\max_a \left(R(s,a) + \sum_{s'} P_{sa}(s') V_{t+1}^*(s') \right)$$

   Make R, P_{sa} be $R^{(t)}, P_{sa}^{(t)}$ for non-stationary problems
   - If make T infinite, value function $V_t^*(s)$ will be unbounded, thus simply make T infinite wouldn't get a discounted MDP formalism (traditional MDP) value iteration, we also need discount γ to ensure a value function bound.

Linear Quadratic Regulation (LQR)

Convenient to develop with the finite horizon setting $\left(\mathcal{S}, \mathcal{A}, P_{sa}^{(t)}, T, R^{(t)}\right)$, but also works with discounted MDP formalism $(\mathcal{S}, \mathcal{A}, P_{sa}, \gamma, R)$

Where to get A, B ?

- Learn from data
  Linear regression on m examples
- Linearize a non-linear model

A remarkable property

- Using LQR make the value function a quadratic function, can solve $V^\star$ exactly.

Dynamic programming for LQR

1. Base case

$$
\begin{aligned}
V_T^\star(s_T) &= \max_{a_T} R\left(s_T, a_T\right) \\
&= -s_T^T U s_T \\
\pi_T^\star(s_T) &= \vec{0}
\end{aligned}
$$

   Initialize $\Phi_T = -U, \Psi_T = \vec{0}$

2. The key step

   It's can be show that

$$
\begin{aligned}
&\text{if } V_{t+1}^*\left(s_{t+1}\right) = s_{t+1}^\top \Phi_{t+1} s_{t+1} + \Psi_{t+1} \\
&\text{then } V_t^*\left(s_t\right) = s_t^\top \Phi_t s_t + \Psi_t
\end{aligned}
$$

   Then we can solve LQR recursively.

   Recursive calculate Φ_t, Ψ_t using Φ_{t+1}, Ψ_{t+1} for $t = T-1, T-2, \ldots, 0$

$$
\begin{aligned}
\Phi_t &= A_t^\top \left(\Phi_{t+1} - \Phi_{t+1} B_t \left(B_t^\top \Phi_{t+1} B_t - W_t\right)^{-1} B_t \Phi_{t+1}\right) A_t - U_t \\
\Psi_t &= -\operatorname{tr}\left(\Sigma_t \Phi_{t+1}\right) + \Psi_{t+1}
\end{aligned}
$$

   Calculate L_t

   $\pi^\star(s_t) = L_t s_t$

$$
\begin{aligned}
L_t &= \left[\left(B_t^\top \Phi_{t+1} B_t - W_t\right)^{-1} B_t \Phi_{t+1} A_t\right] \\
\pi^\star(s_t) = a_t^* &= \left[\left(B_t^\top \Phi_{t+1} B_t - V_t\right)^{-1} B_t \Phi_{t+1} A_t\right] \cdot s_t \\
&= L_t \cdot s_t
\end{aligned}
$$

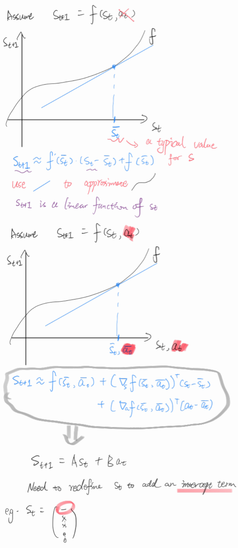
Takeaway: Optimal action is a linear function of state s_t

Fun Fact about LQR

- L_t depends on Φ_{t+1} but not Ψ_{t+1}, means that in order to take action, constant item for the quadratic function doesn't matter. ⚠️ So it's NOT NECESSARY to calculate Ψ in the LQR algorithm

Thus $\pi^\star$, L_t don't depend on Σ_w, but $V^\star$ does.

- Linearize a non-linear model



Statistics

- Probabilistic interpretation (11)

  Assumption: error term are Gaussian and IID

- Posterior Distribution (11.19)

  Every unknown quantity can be estimated by conditioning the model on all the observed quantities. Such a conditional distribution over the unobserved random variables, conditioned on the observed random variables

- model (e.g p (x, y, θ)) (11.19)

  The joint distribution of all the random variables

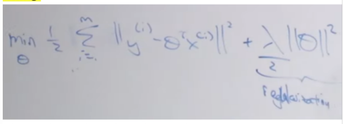
- Prior Distribution (11.19)

  The prior probabilities are to be assigned before we see the data – they need to capture our prior beliefs of what the model parameters might be before observing any evidence, and must be a subjective opinion by the person building the model.

Regularization (63)

$\lambda/2\|\Theta\|^2$

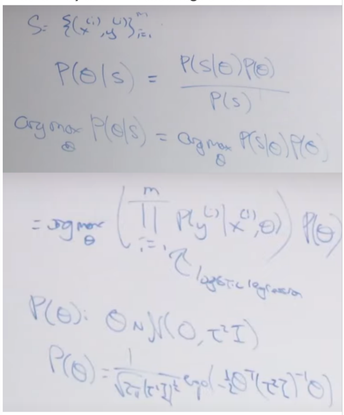
regularization is the most effective way to prevent over-fitting.



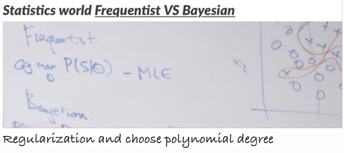
The optimization objective of the support vector machine was to minimize $\|w\|^2$, this turns out to maximize the geometric margin SVM

In order to make sure the λ on the same scale, a common pre-processing step we're using learning algorithms is normalizing different sized features to a similar scale. [the normalization also makes gradient descent run faster]

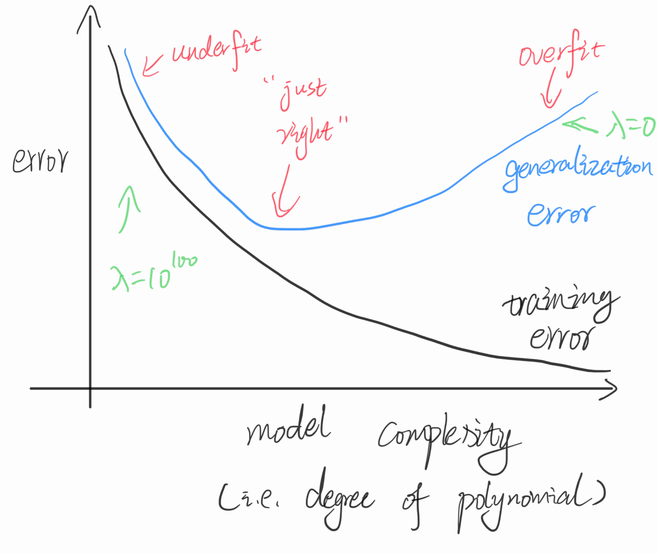
Another way to think about Regularization



Statistics world Frequentist VS Bayesian



Regularization and choose polynomial degree



Cross Validation

82

Different mechanistic procedures to find the optimum point

1. Split your dataset into $S_{train}, S_{dev}, S_{test}$
   S_{dev} also called **Cross Validation Set**
2. Train each model i (option for degree of polynomial) on S_{train}, get some hyposis h_i
3. Measure error on S_{dev}. Pick model with lowest error on S_{dev}
   Don't evaluate algorithms on training set. *Cause over-fit, because more complex algorithm will always do better on the training set*
4. *[Optional]* Evaluate the algorithm on a separate test set (S_{test}) and report that error.

How do you decide how much data should go into $S_{train}, S_{dev}, S_{test}$?

Common Weight on Dataset	S_{train}	S_{dev}	S_{test}
Small dataset(without dev set)	70%	-	30%
Small dataset(with dev set)	60%	20%	20%
Large dataset	90%	5%	5%

- Choose S_{train}, S_{dev} to be big enough
- If you want to tease out very small differences(e.g. 0.001%), you may want a large S_{test} .
- If you want to compare algorithms with large accuracy differences(e.g. 1% vs 2%), small-size S_{test} is enough.

Do not make ANY decisions about your model using the test set.

- Hold-Out Cross Validation (Simple Cross Validation)

82

When you have a large dataset, Cross Validation can be used to choose

1. the model of polynomial
2. the regularization parameter λ or C or τ

- K-Fold Cross Validation

83

- Makes more efficient of the data
- Computationally expensive

When you have a small dataset, without too much data waste

$K = 10$ is typical

[Use when $m = 50$(roughly) or less] When $K = m$, this method called **Leave-one-out Cross Validation**

- Feature Selection 84

  A special case for model selection

  If you have a lot of features, one way to reduce over-fitting is to try to find a small subset of most useful features for your task.

  Forward Search

  Keep iterating until adding more features now hurts performance, then pick whichever feature subset allows you to have the best possible performance of dev set.

  Backward Search

- Value Iteration VS Policy iteration

  Value Iteration

  Policy Iteration

  For small MDPs, policy iteration is often very fast and converges with very few iterations. However, for MDPs with large state spaces, solving for Vπ explicitly would involve solving a large system of linear equations, and could be difficult. In these problems, value iteration may be preferred.

  Value iteration VS Policy Iteration

  - Policy iteration works good for relatively small `#states` problems, poor on large.
    For small problem, policy iteration converge faster than value iteration.
  - Value Iteration will converge to V^*, but won't ever get to exactly V^*

Modeling Strategies

- Error Analysis 177

  knowing which parts of the machine learning algorithm lead to this error or score
  tries to explain the difference between current performance and perfect performance

- Ablative Analysis 179

  tries to explain the difference between some baseline (much poorer) performance and current performance.

- cs229-ps
- cs229-ps_sol
- Machine Learning Yearning
- 机器学习 Machine Learning
- 图解机器学习
- cs229-extra_notes

- Group
  - cs229-notes
    - ML Terminology
      - Classification
        - Unsupervised Learning
    - Mathematical Methods
      - Algorithms
    - Lectures
      - Lecture 2
        - Supervised learning 1

          Linear Regression
        - Linear Regression 3

          - Linear Regression has no local optimum

          Parametric Learning Algorithms

          Supervised learning
        - normal equations 11

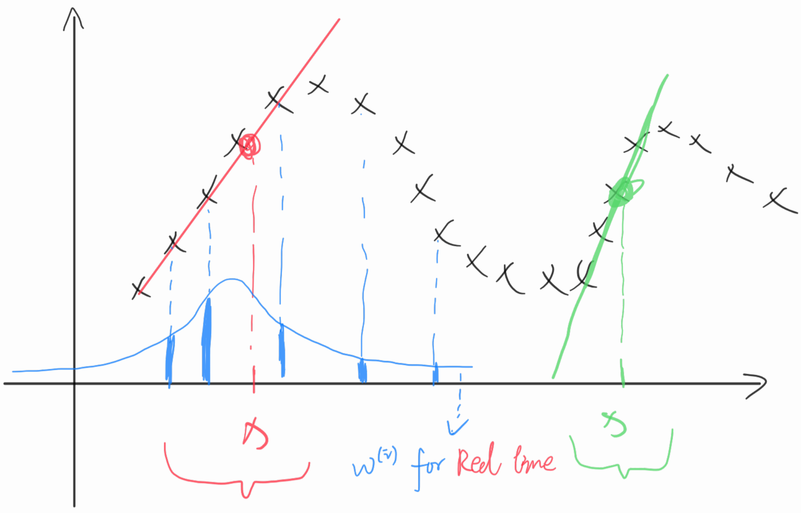
          $$X^T X\theta = X^T\vec{y}$$

          $$\theta = (X^T X)^{-1} X^T \vec{y}.$$

          $$X^T X\theta = X^T\vec{y}$$
          $$\theta = (X^T X)^{-1} X^T\vec{y}$$

          #Formula
        - Locally weighted linear regression 13

          Tend to use: when you have a relatively low dimensional dataset (i.e. #features is not too big) and have a lot of data, and don't wanna think about what features to use.

          1. Fit θ to minimize $\sum_i w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$.
          2. Output $\theta^T x$.

          $$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

          The choice of the bandwidth parameter τ has an effect on overfitting and underfitting.
          - Too big end up over-smoothing the data
          - Too small end up fitting a very jagged fit to the data

          Relative slow, but good enough for small dataset (up to thousands), while dataset size is too big (say, millons), use KD tree etc is better.

          #Formula

          Non-Parametric Learning Algorithms

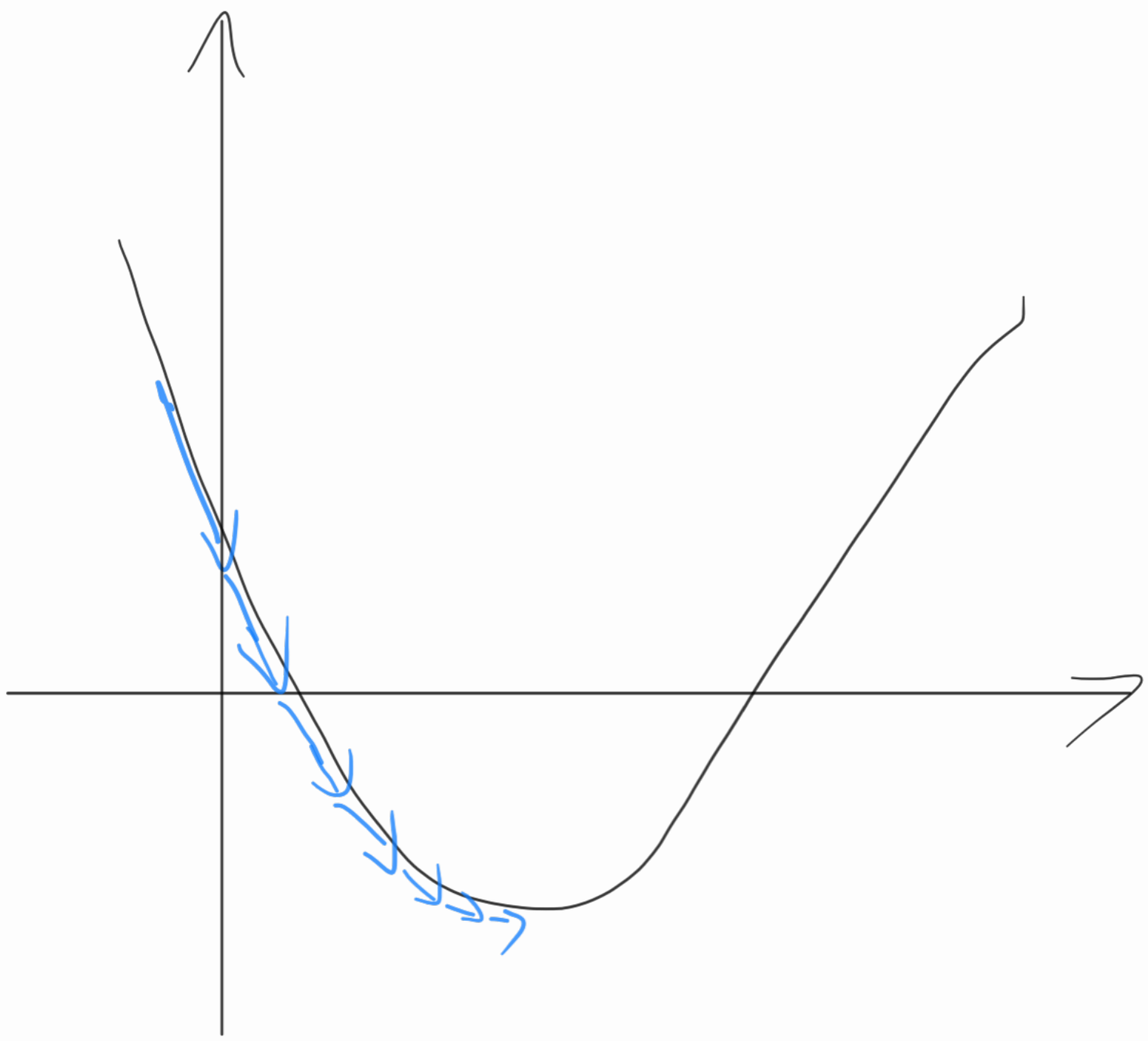
LMS algorithm

4

LMS stands for “least mean squares”

gradient descent

4



- batch gradient descent

5

> Repeat until convergence {
>
> $$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}.$$
>
> }

looks at every example in the entire training set on every step

- stochastic gradient descent

7

> Loop {
>
> for i=1 to m, {
>
> $$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}.$$
>
> } using the derivative of just one single example
>
> }

When the size of training dataset is too large, using stochastic gradient descent.

stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at.

Lecture 3

- Probabilistic interpretation 11

  Assumption: error term are Gaussian and IID

- Parametric Learning Algorithms

  Fit some fixed of parameters (**θ_i**) to data

  Linear Regression

- Non-Parametric Learning Algorithms

  Amount of data/parameters you need to keep grows (linearly) with the size of dataset.

  - Need to keep all the data in computer memory to make predications.

  Locally weighted linear regression

▼ Classification and logistic regression 16

- binary classification 16

▼ Logistic regression 16

Hypothesis

$$h_\theta(x) = g\left(\theta^T x\right) = \frac{1}{1 + e^{-\theta^T x}}$$

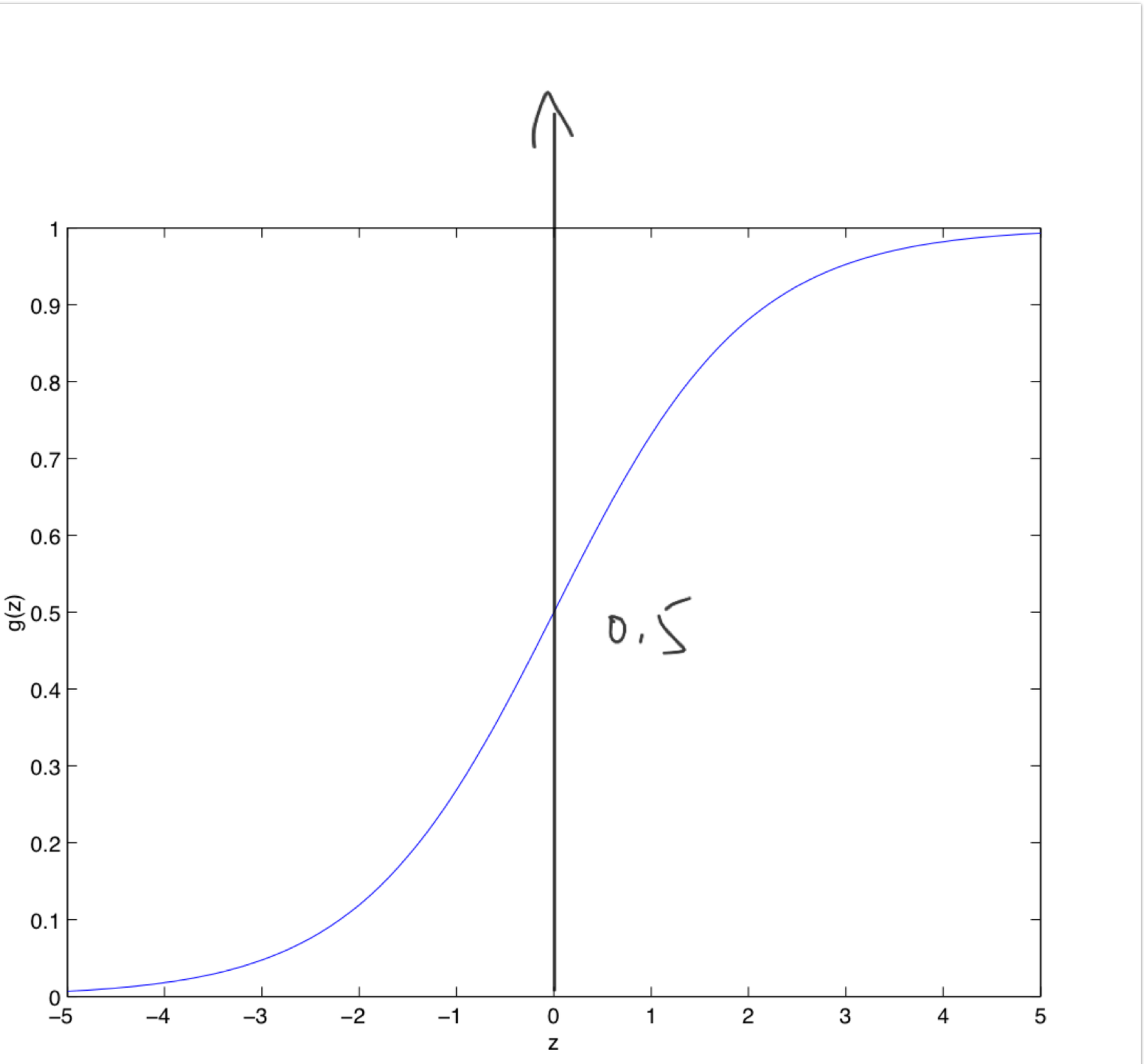
Logistic/Sigmoid Function

$$g(z) = \frac{1}{1 + e^{-z}}$$

By convention,

$$p(y \mid x; \theta) = (h_\theta(x))^y \left(1 - h_\theta(x)\right)^{1-y}$$

g(z) demenstration



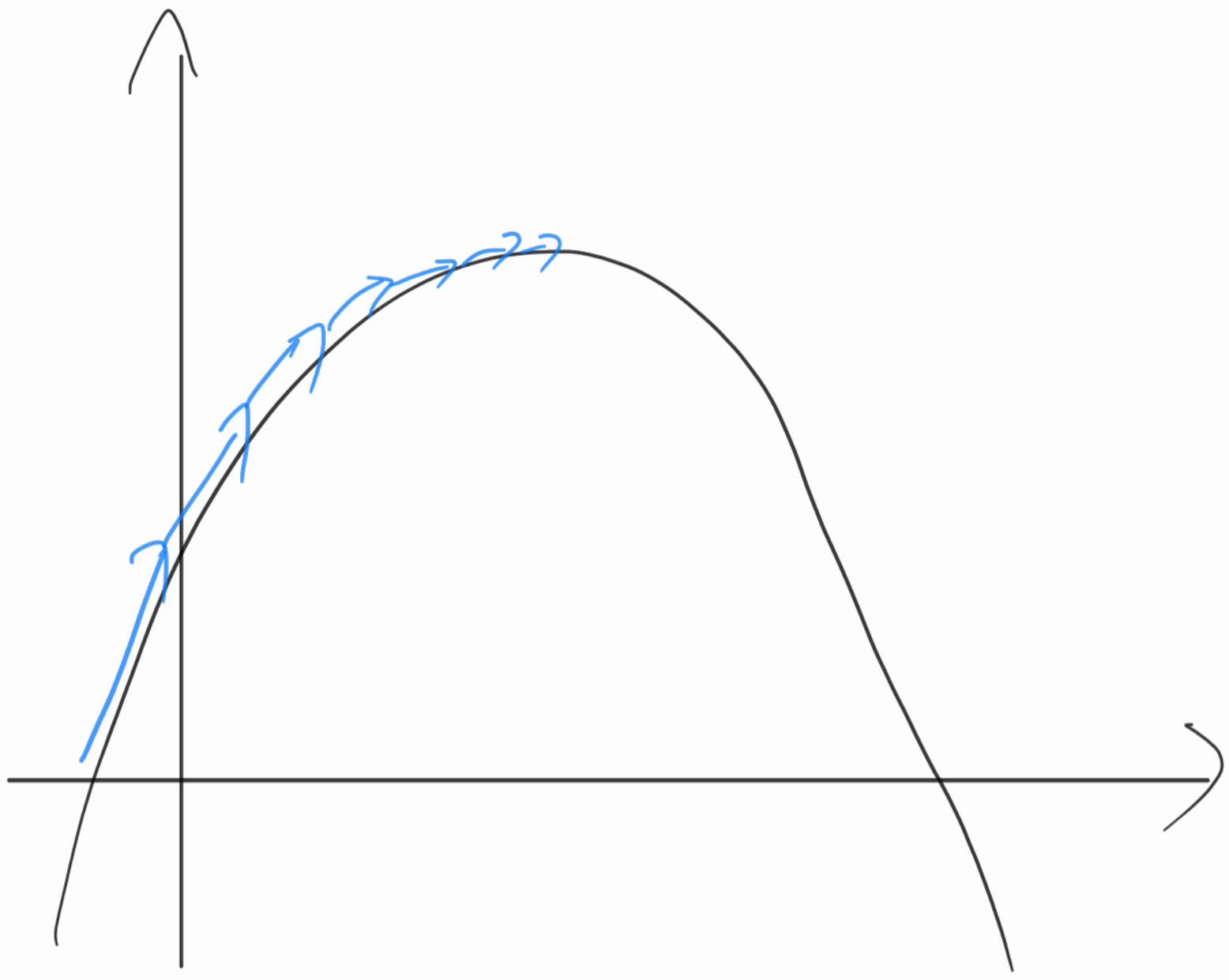
UPDATE RULES

#Formula

gradient ascent

Newton’s method

gradient ascent



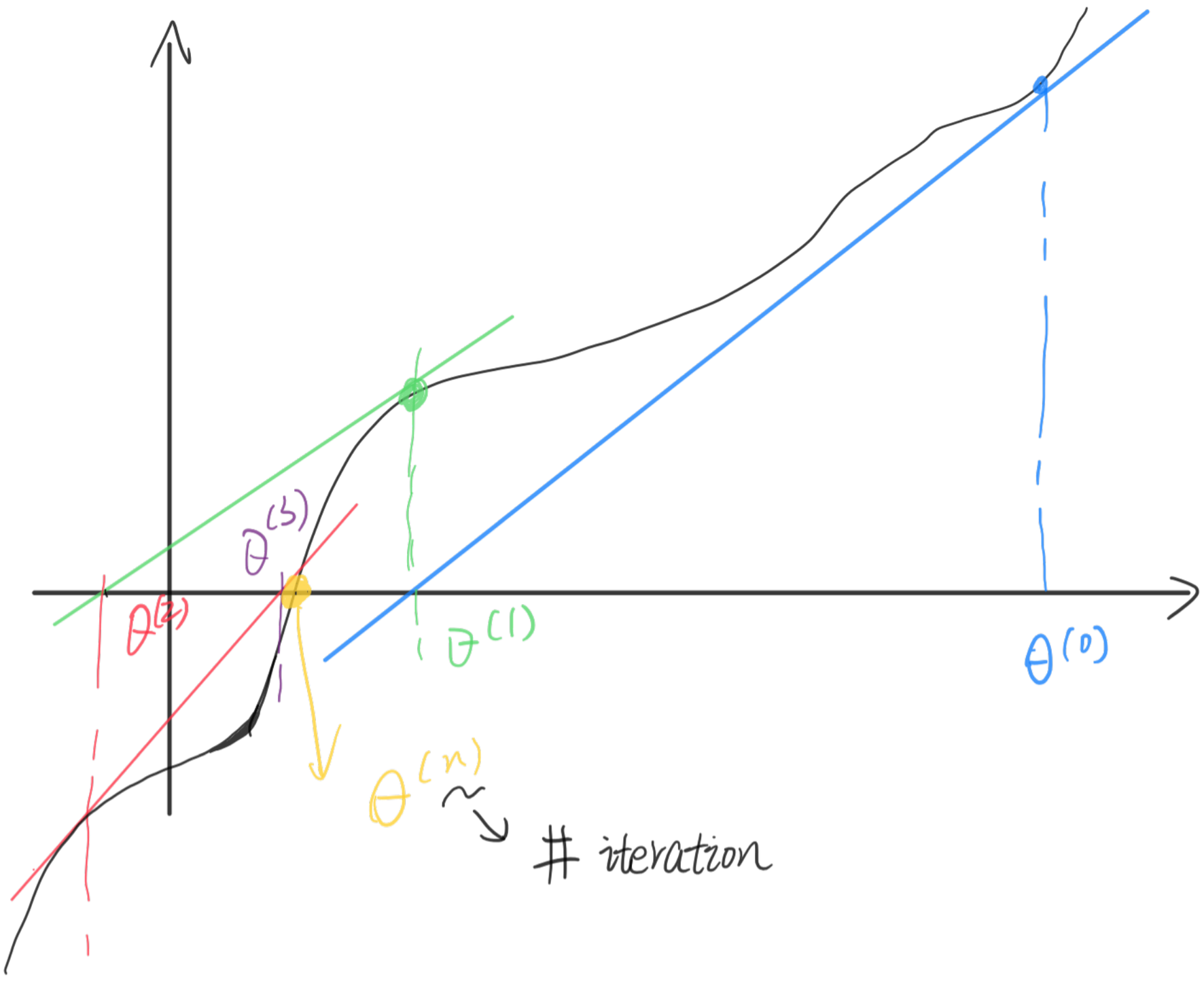
- gradient ascent 18

$$\theta_j := \theta_j + \alpha \sum_{j=1}^{m} \left(y^{(i)} - h_\theta \left(x^{(i)} \right) \right) x_j^{(i)}$$

- stochastic gradient ascent 18

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta \left(x^{(i)} \right) \right) x_j^{(i)}$$

- Newton's method 20



When Θ is a real number

$$For\ 1st\ \theta, \theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

$$For\ its\ predecessor, \theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$

When Θ is a vector

$$\theta := \theta - H^{-1} \nabla_\theta \ell(\theta)$$

$$H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}$$

Newton's Method has quadratic convergence property.

PROS:
Newton's method needs relatively fewer iteration steps (compared with Gradient ascent) to converge.
CONS:
In high-dimensional problems, if θ is a vector, each step of Newton's Method oils much more expensive.

Lecture 4

- perceptron

"Hard mode" of sigmoid function

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

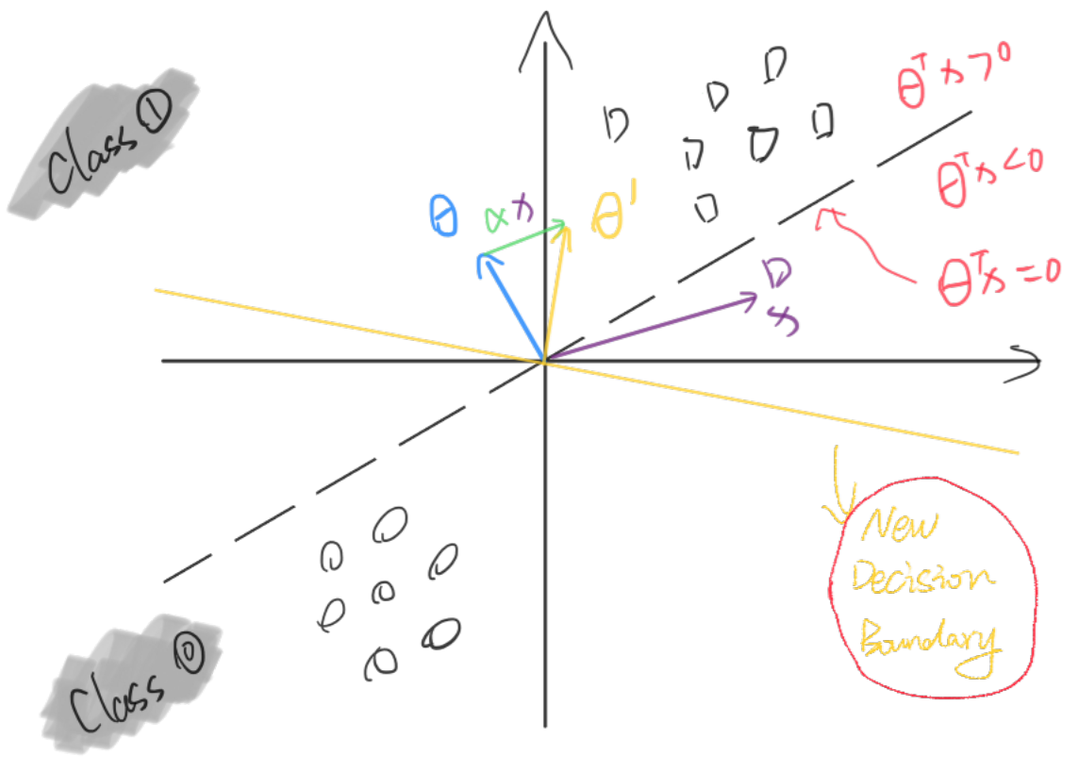
UPDATE RULE

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta \left(x^{(i)} \right) \right) x_j^{(i)}$$

$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)}))\, x_j^{(i)}$

$\cup = \begin{cases} 0 & \text{if algo is right} \\ +1 & \text{if wrong \& } y^{(i)} = 1 \\ -1 & \text{if wrong \& } y^{(i)} = 0 \end{cases}$

Wanted:
$\theta \approx x \mid y=1$
$\theta \not\approx x \mid y=0$



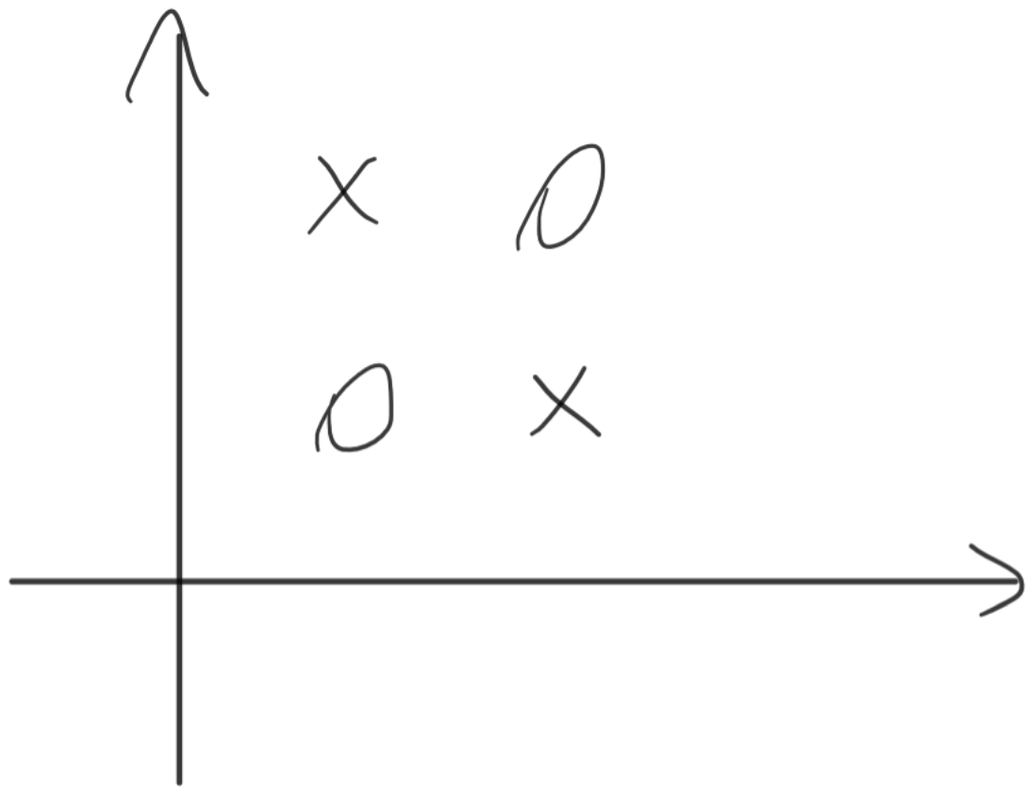
D — Newly added example, which is misclassified

When D is added, $y^{(i)} - h_\theta(x^{(i)}) = 1$

$(1 - 0 = 1)$

#Tips
If you want a vector ***A*** more similar to vector ***B*** , *you can simply add* ***B*** to ***A***

Cons:
- Cannot be used in practice, because perceptron doesn't't have probabilistic interpretation of what's happening, just have a geometrical feel
- Cannot deal with the following situation



- exponential family

22

a class of distributions is in the exponential family if it can be written in the form
(e.g. Bernoulli distribution is a member of the exponential family)

$$p(y;\eta) = b(y)\exp\left(\eta^T T(y) - a(\eta)\right)$$

Features:

① MLE with respect to η is concave;
Maximum Likelihood Estimation
NLL is convex

② $E(y;\eta) = \frac{\partial}{\partial \eta} a(\eta)$

③ $Var(y;\eta) = \frac{\partial^2}{\partial \eta^2} a(\eta)$

When you want E and Var for a probability distribution, only need to differentiate, while generally integrate.

Some common exponential distribution:

Real — Gaussian
Binary — Bernoulli
Count — Poisson
R^+ — Gamma, Exponential
Distribution — beta, Dirichlet [Bayesian statistics]

- Generalized Linear Models

22

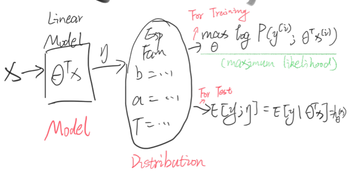
No matter what kind of GLM/which choice of distribution that you make, **the learning update rule is the same, just need different** $h_\theta(x^{(i)})$

$$\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right)x_j^{(i)}$$

Assumptions / Design Choices

1) $y|x;\theta \sim$ Exponential Family (η)
2) $\eta = \theta^T x \quad \theta \in \mathbb{R}^n \quad x \in \mathbb{R}^n$
3) Test time: output $E[y|x;\theta]$
$\Rightarrow h_\theta(x) = E[y|x;\theta]$

The Big Picture for GLM



b, a, T based on the distribution of your choice
(Classification, regression...)

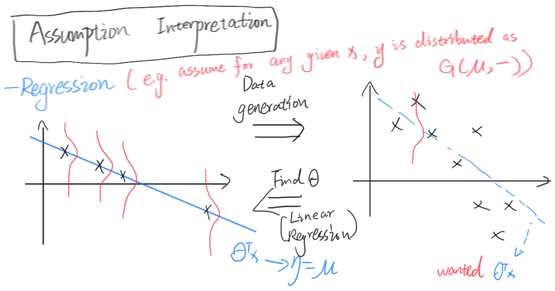
Training θ to predict the parameter of the exponential family distribution whose mean is the prediction that we gonna make for y

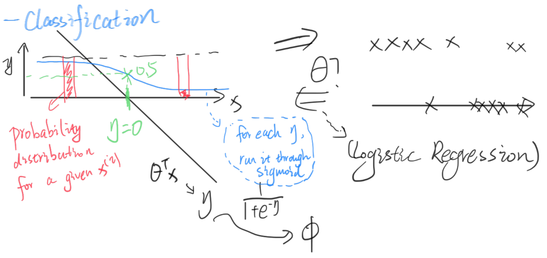
Terminology
natural parameter — η
canonical response function — $g(\eta) = \mathbf{E}[T(y);\eta]$
canonical link function — g^{-1}

$$g(\eta) = E[T(y);\eta] = \frac{\partial}{\partial \eta} a(\eta)$$

(Ref. Exponential Family features)

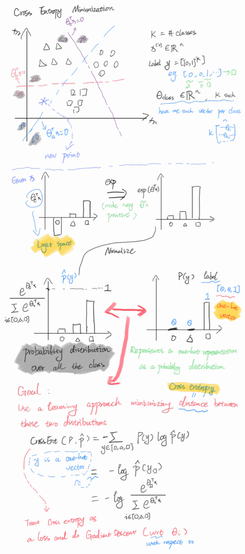
GLMs are just a general way to model data (as long as you have a distribution that can model that kind of data and falls in exponential family)





- Softmax Regression

26



Lecture 5

Generative learning algorithms
- Gaussian Discriminant Analysis (GDA)
- Generative & Discriminative comparison
-Naive Bayes

Generative Learning algorithms

31

- Gaussian discriminant analysis (GDA)

32

classification problem in which the input features x are continuous-valued random variables
GDA is computationally efficient

Modeling

The GDA model is :

$$y \sim \text{Bernoulli}(\phi)$$
$$x \mid y = 0 \sim \mathcal{N}(\mu_0, \Sigma)$$
$$x \mid y = 1 \sim \mathcal{N}(\mu_1, \Sigma)$$

The log-likelihood is :

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \log \prod_{i=1}^{m} p\left(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma\right)$$
$$= \log \prod_{i=1}^{m} p\left(x^{(i)} \mid y^{(i)}; \mu_0, \mu_1, \Sigma\right) p\left(y^{(i)}; \phi\right)$$

Maximum Likelihood Estimation :

$$\phi = \frac{1}{m} \sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}$$
$$\mu_0 = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\} x^{(i)}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\}}$$
$$\mu_1 = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\} x^{(i)}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}}$$
$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)} - \mu_{y^{(i)}}\right)\left(x^{(i)} - \mu_{y^{(i)}}\right)^T$$

Prediction

Goal : output what you think of y for an given x, ***rather than put put a probability*** .

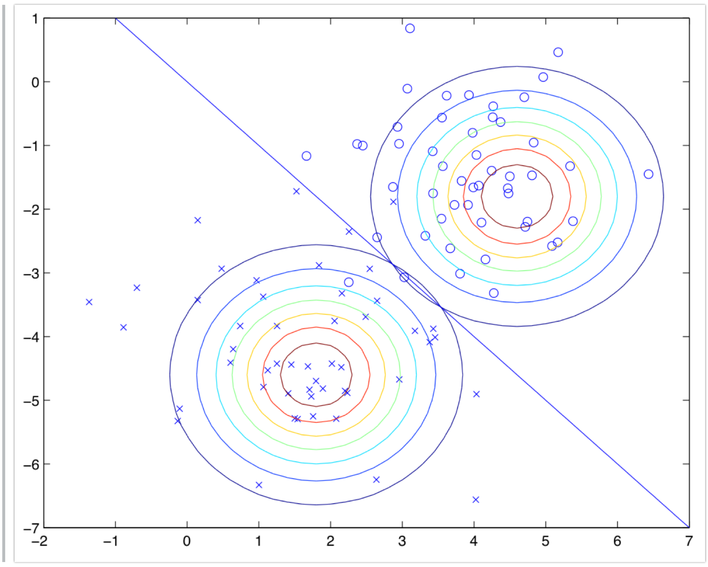
$$\arg\max_y p(y \mid x) = \arg\max_y \frac{p(x \mid y)p(y)}{p(x)}$$
$$= \arg\max_y p(x \mid y)p(y)$$

* FYI

arg max/min is the value you need plug in to achieve (i.e. argument) that max/min value.

e.g. $\min (x-5)^2 = 0$

$\arg\min (x-5)^2 = 5$



use same covariance matrix Σ (for y=0 and y=1), the decision end up being linear, different Σ is also okay to build up decision boundary, except it won't be linear anymore.

Naive Bayes

38

Very efficient (slower than Logistic Regression though) and also keep on updating even as you get new data.

SCENARIO: face a problem and implement something quick and dirty to solve it

Naive Bayes (NB) assumption

assume that the xi's are conditionally independent given y.

Joint Likelihood

$$\mathcal{L}\left(\phi_y, \phi_{j|y=0}, \phi_{j|y=1}\right) = \prod_{i=1}^{m} p\left(x^{(i)}, y^{(i)}\right)$$

MLE

$$\phi_{j|y=1} = \frac{\sum_{i=1}^{m} 1\left\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\right\}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}}$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^{m} 1\left\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\right\}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\}}$$

$$\phi_y = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}}{m}$$

PROS:
- Computationally efficient
- Relatively quick to implement
- Doesn't require iterative

CONS:
- Logistic regression work better in terms of delivering and higher accuracy than Naive Bayes

- Laplace Smoothing

41

$$\phi_j = \frac{\sum_{i=1}^{m} 1\left\{z^{(i)} = j\right\} + 1}{m + k}$$

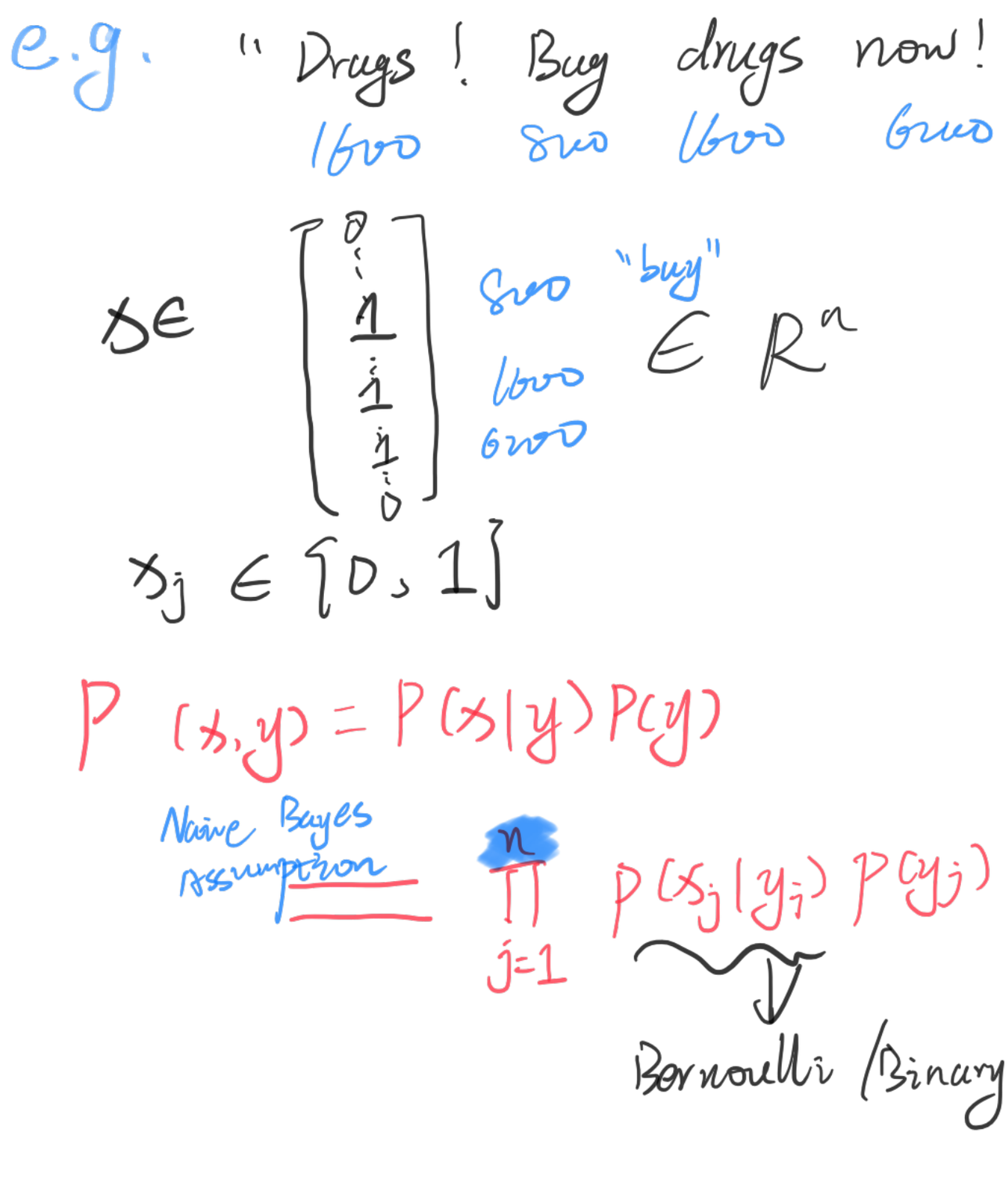
z taking values in {1, . . . , k}
For Laplace smoothing, we add 1 to the numerator, and k to the denominator

Event Models

43

- Multi-Variate Bernoulli Event Model

43



- Multinomial Event Model

43

e.g. "Drugs! Buy drugs now!"
1600 800 1600 6000

$$x \in \begin{bmatrix} 1600 \\ 800 \\ 1600 \\ 6000 \end{bmatrix} \in R^n$$

$x_j \in \{1, 2, \cdots 10000\}$

n_i = length of email

$$P(x, y) = P(x|y)P(y)$$

Naive Bayes assumption

$$\prod_{j=1}^{n} P(x_j|y_i)P(y_i)$$

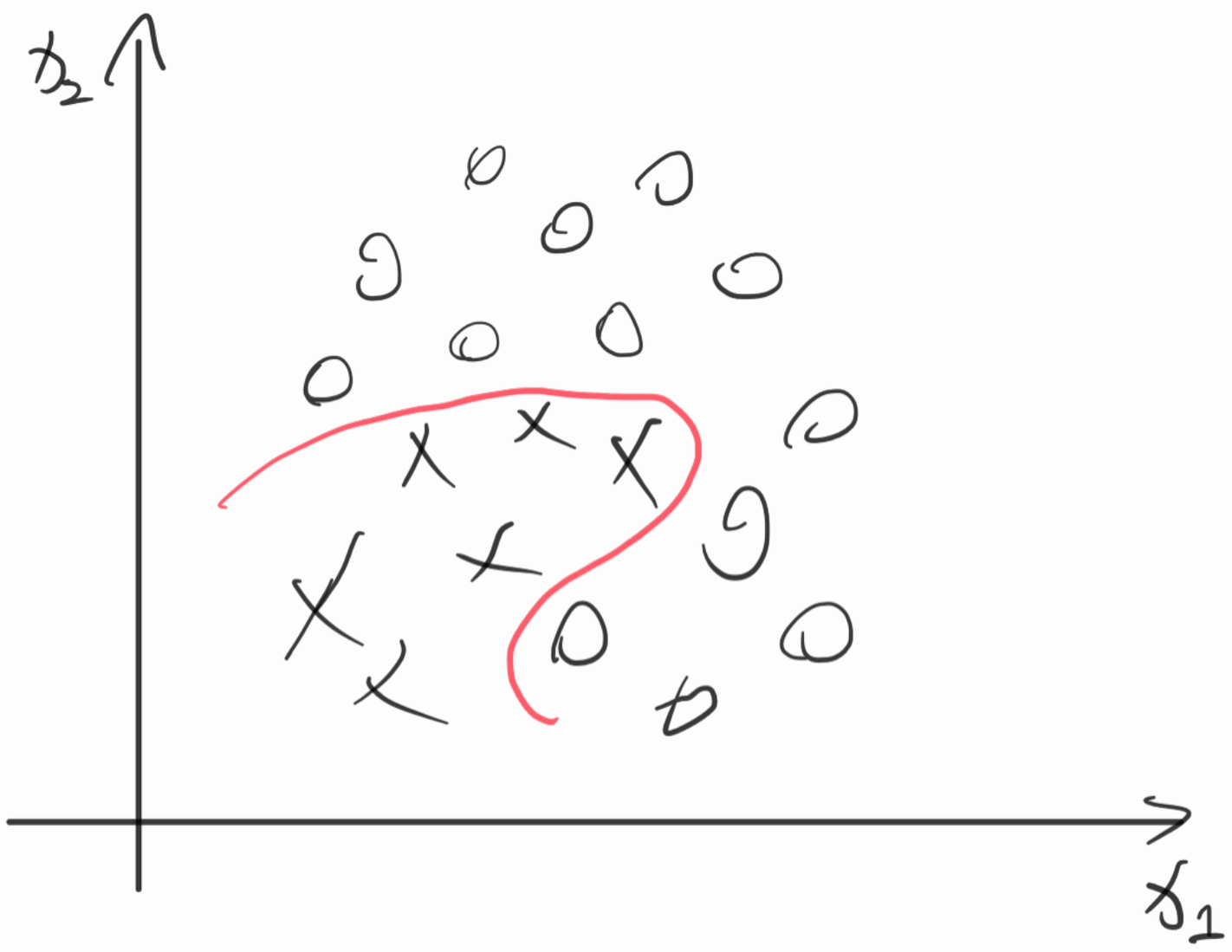
Lecture 6 Supportive Vector Machine

Support Vector Machines

45

SVM = Optimal Margin Classifier + Kernel tricks

Find a hyperplane in a high-dimension space and project that decision hyperplane into a (non-linear) decision boundary in your low-dimension space.



PROS
- Turn-key, (i.e. doesn't have many parameters to fiddle with)

CONS
- not as effective as neural network

Notation

$$y \in \{-1, 1\}$$
$$h_{w,b}(x) = g\left(w^T x + b\right)$$
$$g(z) = 1 \text{ if } z \geq 0$$
$$g(z) = -1 \text{ otherwise}$$

Due to $g(z)$, classifier will directly predict either -1 or 1.

Optimal Margin Classifiers

54

Basic building block for the supportive vector machine.

Optimal Margin Classifier is the algorithm tries to maximize the geometric margin.

$$\min_{\gamma,w,b} \frac{1}{2}\|w\|^2$$
$$\text{s.t. } y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1, \quad i = 1, \ldots, m$$

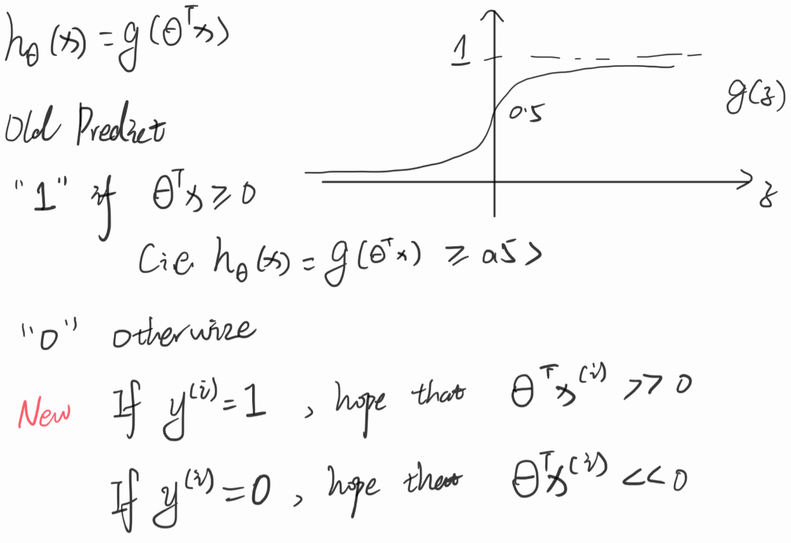
Dual Optimization Problem

$$max_\alpha\, W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \left\langle x^{(i)}, x^{(j)} \right\rangle$$
$$\text{s.t. } \alpha_i \geq 0, \quad i = 1, \ldots, m$$
$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

- Functional Margin

47

How confidently and accurately do you classify an example



Functional Margin

$$\hat{\gamma}^{(i)} = y^{(i)}\left(w^T x + b\right)$$
$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)}$$

Attention

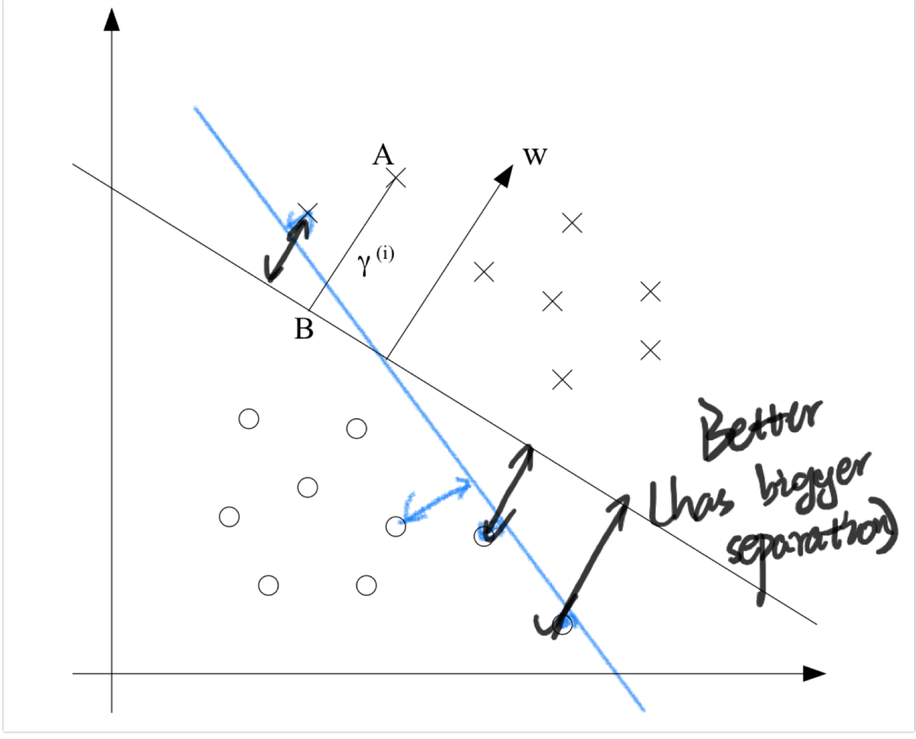
$h_{w,b}(x) = g\left(w^T x + b\right)$ depend only on the signal but not magnitude of $w^T + b$– **Use normalization to solve this situation**

Normalization

$$\text{replace } (w, b) \text{ with } (w/\|w\|_2, b/\|w\|_2)$$

- Geometric Margin

48



Geometric Margin

$$\gamma^{(i)} = y^{(i)}\left(\left(\frac{w}{\|w\|}\right)^T x^{(i)} + \frac{b}{\|w\|}\right)$$
$$\gamma = \min_{i=1,\ldots,m} \gamma^{(i)}$$

if $\|w\| = 1$, geometric margin is equal to functional margin.

- Geometric Margin is invariant to rescaling parameters.

- Kernels

57

Map low-dimensional feature vector into high-dimensional(maybe infinite) set of feature vector.

PROS

- Relieves us from manually picking features

Kernel Trick:

1. Write algorithm in terms of $< x^{(i)}, x^{(j)} >$ (or $< x, z >$)
2. Let there be some mapping from your original input features $x \rightarrow \phi(x)$
3. Find way to compute $K(x,z) = \phi(x)^T \phi(z)$ – **Kernel Function**
4. Replace $< x, z >$ in algorithm with $K(x,z)$

If you doing this replace, you 're running the whole learning algorithm on high dimensional set of features

Common Kernel Functions

Gaussian Kernel *(Mostly used)*

$$K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$$

Polynoimal Kernel

$$K(x,z) = (x^T z)^d$$

- Non-Separable Case

63

- L1 Regularization

63

> l1 regularization

$$min_{\gamma,w,b}\ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i$$

$$\text{s.t. } y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \xi_i, \quad i = 1, \ldots, r$$

$$\xi_i \geq 0, \quad i = 1, \ldots, m.$$

Dual form for L_1 Norm Soft Margin SVM

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j \left\langle x^{(i)}, x^{(j} \right.$$

$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m$$

$$\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$$

make the algorithm work for non-linearly separable datasets as well as be less sensitive to outliers

- L2 Regularization

173

- Lecture 7 Kernels

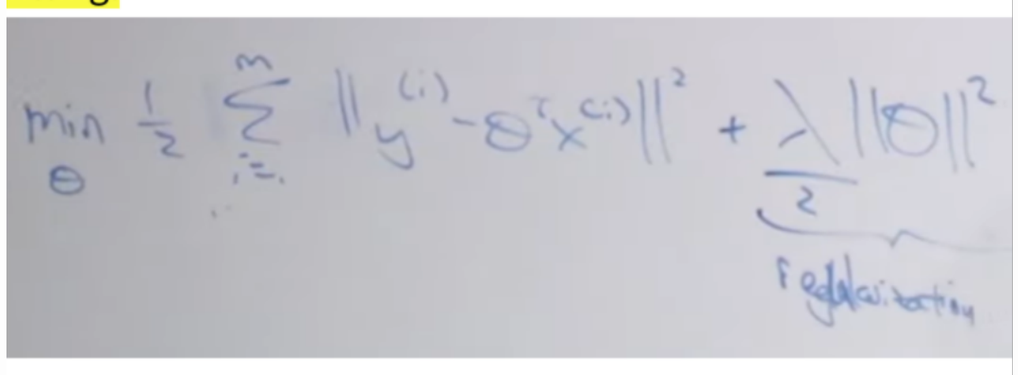
  - Optimization problem
  - Representer theorem
  - Kernels
  - Examples of kernels

Lecture 8

Regularization

$\lambda/2\|\Theta\|^2$

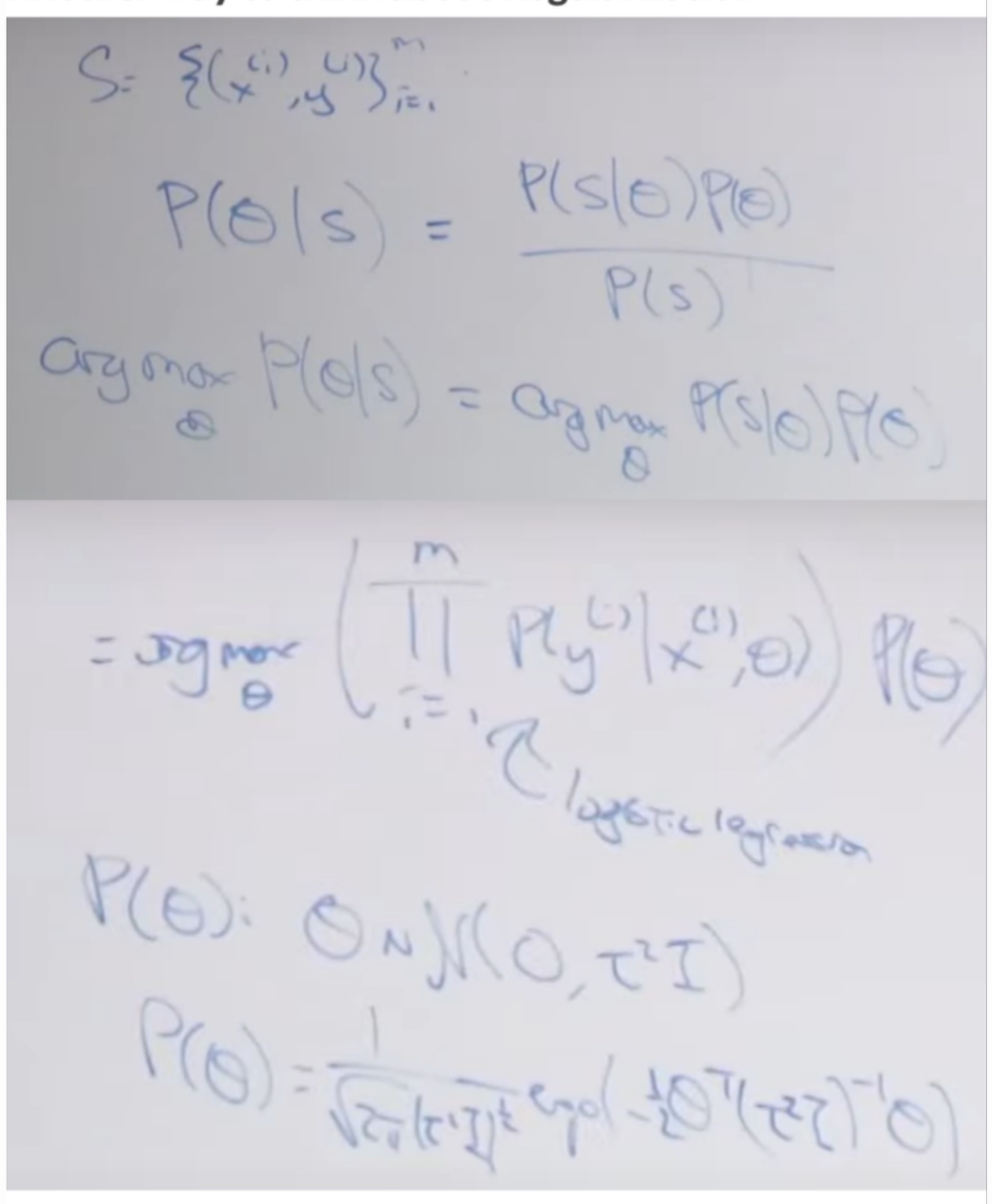
regularization is the most effective way to prevent over-fitting.



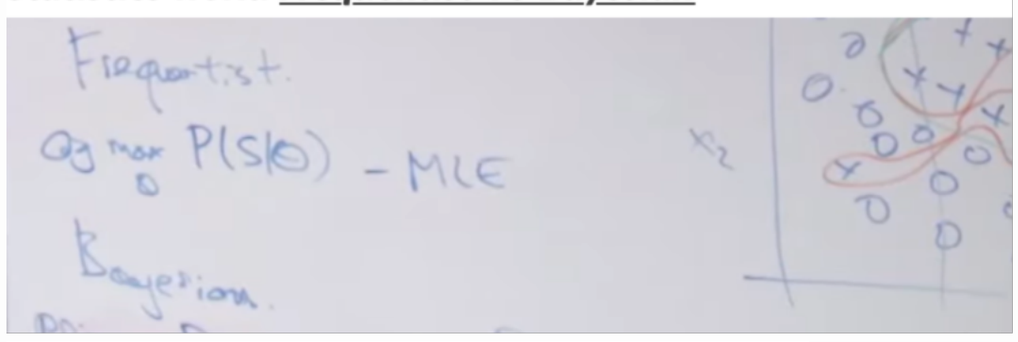
The optimization objective of the support vector machine was to minimize $\|w\|^2$, this turns out to maximize the geometric margin SVM

In order to make sure the λ on the same scale, a common pre-processing step we're using learning algorithms is normalizing different sized features to a similar scale. [the normalization also makes gradient descent run faster]

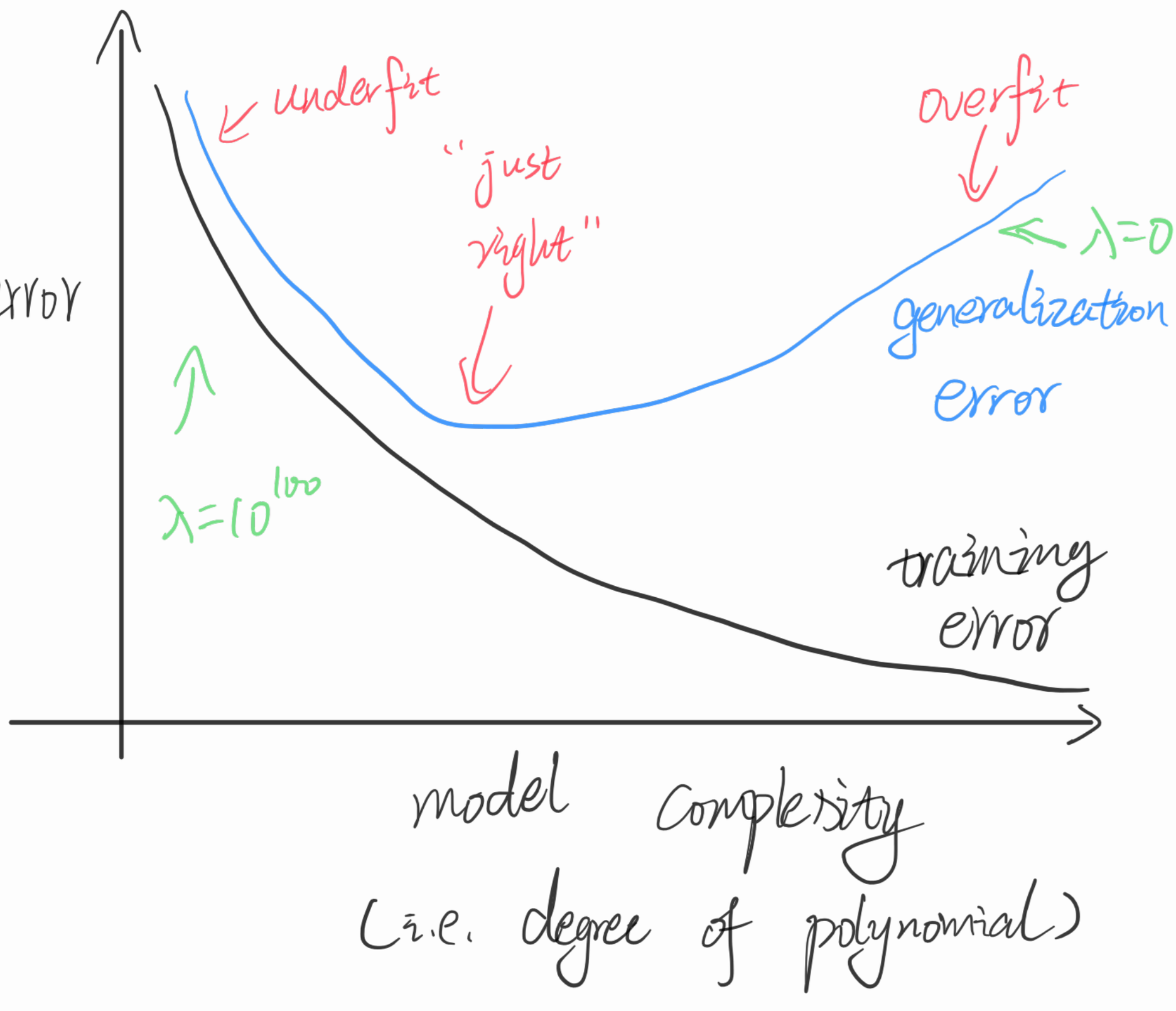
Another way to think about Regularization



Statistics world Frequentist VS Bayesian



Regularization and choose polynomial degree



Cross Validation

82

Different mechanistic procedures to find the optimum point

1. Split your dataset into $S_{train}, S_{dev}, S_{test}$
   S_{dev} also called **Cross Validation Set**
2. Train each model i (option for degree of polynomial) on S_{train}, get some hyposis h_i
3. Measure error on S_{dev}. Pick model with lowest error on S_{dev}
   Don't evaluate algorithms on training set. *Cause over-fit, because more complex algorithm will always do better on the training set*
4. *[Optional]* Evaluate the algorithm on a separate test set (S_{test}) and report that error.

How do you decide how much data should go into $S_{train}, S_{dev}, S_{test}$?

Common Weight on Dataset	S_{train}	S_{dev}	S_{test}
Small dataset(without dev set)	70%	-	30%
Small dataset(with dev set)	60%	20%	20%
Large dataset	90%	5%	5%

- Choose S_{train}, S_{dev} to be big enough
- If you want to tease out very small differences(e.g. 0.001%), you may want a large S_{test} .
- If you want to compare algorithms with large accuracy differences(e.g. 1% vs 2%), small-size S_{test} is enough.

Do not make ANY decisions about your model using the test set.

- Hold-Out Cross Validation (Simple Cross Validation)

82

When you have a large dataset, Cross Validation can be used to choose

1. the model of polynomial
2. the regularization parameter λ or C or τ

- K-Fold Cross Validation

83

  - Makes more efficient of the data
  - Computationally expensive

  When you have a small dataset, without too much data waste

  $K = 10$ is typical

  [Use when $m = 50$(roughly) or less] When $K = m$, this method called **Leave-one-out Cross Validation**

- Feature Selection

  A special case for model selection

  If you have a lot of features, one way to reduce over-fitting is to try to find a small subset of most useful features for your task.

  Forward Search

  Keep iterating until adding more features now hurts performance, then pick whichever feature subset allows you to have the best possible performance of dev set.

  Backward Search

Lecture 9 Learning Theory

- Setup/Assumptions
- Bias & Variance
- Approximation Estimate
- Empirical Risk Minimize
- Uniform Convergence
- VC dimension

Approx/Estimation Error &ERM

Empirical Risk Minimization (ERM)

73

$$\hat{\varepsilon}(h) = \frac{1}{m}\sum_{i=1}^{m} 1\left\{h\left(x^{(i)}\right) \neq y^{(i)}\right\}$$

$$\hat{\theta} = \arg\min_{\theta} \hat{\varepsilon}\left(h_{\theta}\right)$$

- Hoeffding Inequality

74

$$P\left(\left|\varepsilon\left(h_i\right) - \hat{\varepsilon}\left(h_i\right)\right| > \gamma\right) \leq 2\exp\left(-2\gamma^2 m\right)$$

Lecture 10 Decision Trees & Ensembling

Decision Trees

(6) 122

- Cross Entropy Loss

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{k} y_j \log \hat{y}_j$$

- Ensembling

Ensembling

Take X_i's which are random variables that are independent identically distributed (IID)

$$Var(X_i) = \sigma^2$$

$$Var(\bar{X}) = Var(\frac{1}{n}\sum_i X_i) = \frac{\sigma^2}{n}$$

Drop the independence assumption, so now X_i's just identically distributed(ID), X's correlated by ρ.

$$Var(\bar{X}) = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

Ways to Ensemble

1. ~~different algorithms~~
2. ~~different training sets~~
3. Bagging (*e.g. Random Forests*)
   Try to approximate having different training sets.
4. Boosting (*e.g. AdaBoost, XGBoost*)

Bagging - Bootstrap Aggregation

Take a bunch of bootstrap samples, train separate models on each and then average their outputs.

Bootstrap

1. Have a true population P
2. Training set $S \sim P$
3. Assume $P = S$
4. Bootstrap samples $Z \sim S$

Bootstrap Aggregation

1. Bootstrap samples $Z_1, Z_2, Z_3..., Z_M$
2. Train model G_m on Z_m
3. Define a meta model $G(m) = \frac{\sum_{m=1}^{M} G_m(x)}{M}$

Bias-Variance Analysis

$Var(\bar{X}) = \rho\sigma^2 + \frac{1-\rho}{M}\sigma^2$

- Bootstrapping is driving down ρ
  Increasing the number of bootstrap models in your training, doesn't cause you to overfit anymore than you were beforehand.
- More M leads to less variance. *(There is a lower bound, can't make variance 0)*
- Bias is slightly increased because of random subsampling. *(Because the bootstrap samples Z are actually subsets of the original set S, so you model become less complex, that increases bias)*

Decision trees + Bagging

Decision trees are high variance, low bias.

Ideal fit for bagging

Random Forests

At each split, consider only a fraction of total features.

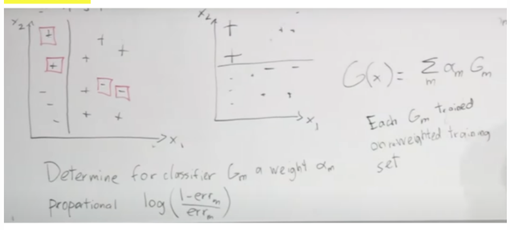
- Decrease ρ
- Decorrelate Models

Boosting

Adaboost, XGBoost, gradient boost machine

- Decrease bias
- Additive

AdaBoost



Lecture 11 Introduction to Neural Networks

DeepLearning

- Xavier/He Initialization 170

$$w^{[\ell]} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n^{[\ell]} + n^{[\ell-1]}}}\right)$$

Lecture 12 Backpropagation & Improving Neural Networks

Lecture 13 Debugging ML Models & Error Analysis

Syllabus:

1. Diagnostics for debugging learning algorithms
2. Error analyses and ablative analysis
3. How to get started in a machine learning problem.
    - Premature (statistical) optimization

- Error Analysis 177

  knowing which parts of the machine learning algorithm lead to this error or score
  tries to explain the difference between current performance and perfect performance

- Ablative Analysis 179

  tries to explain the difference between some baseline (much poorer) performance and current performance.

Lecture 14 EM algorithms

Expectations-Maximization 95

Solve MLE directly might be hard(Because z is a latent variable). Instead, EM repeatedly construct a lower-bound on l (E-step), and then optimize that lower-bound (M-step).

EM implements a softer way of assigning points to the different cluster centroids.

- To do EM, we need a concave function

1. **E-step**

$$Q_i(z^{(i)}) = P(z^{(i)} \mid x^{(i)}; \theta)$$

$Q_i(z^{(i)})$ *is* $w_j^{(i)}$

2. **M-step**

$$\theta := \arg\max_\theta \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)};}{Q_i(z^{(i)})}$$

Iterative E-step and M-step, the algorithm should converge to a local optima.

Question: Why don't we just take $\arg\max_\theta l(\theta)$ *?*

Because there's no known way to solve that.

Repeat until convergence: {

(E-step) For each i, j, set w ^{(i)} j is how much x ^{(i)} is assigned to \mu j Gaussian.

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

(M-step) Update the parameters:

$$\begin{aligned}
\phi_j &:= \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)}, \\
\mu_j &:= \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}}, \\
\Sigma_j &:= \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}
\end{aligned}$$

}

$$p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{\sum_{l=1}^{k} p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)}$$

- The K-Means Clustering Algorithm

1. Initialize **cluster centroids** $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$ randomly.
2. Repeat until convergence: {

"Color the points"

For every i, set

$$c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$$

"Moves the cluster"

For each j, set

$$\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}}.$$

}

How to choose K ?

Issue: `#clusters` might be ambiguous.

Solution:

1. AIC or BIC criteria for automatically choosing `#clusters`
2. **Choose manually**

What to do when K-means stuck in local minima?

Run K-means on different iteration times and different initializations of cluster centroids. Pick the lowest cost function $J(c, \mu)$ run.

- Mixture of Gaussians

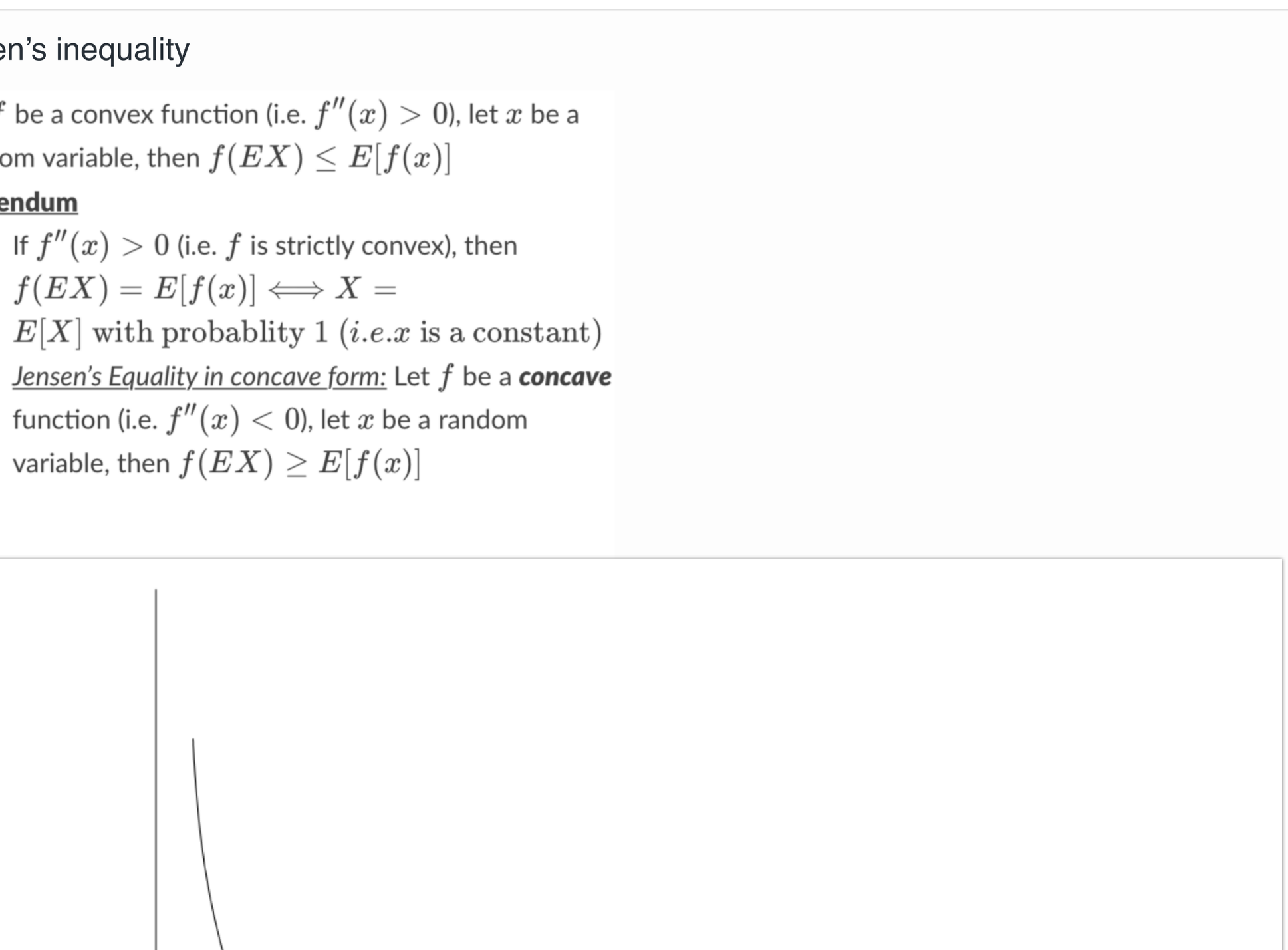
model posits that each x(i) was generated by randomly choosing z(i) from {1, . . . , k}, and then x(i) was drawn from one of k Gaussians depending on z(i).

- Jensen's inequality

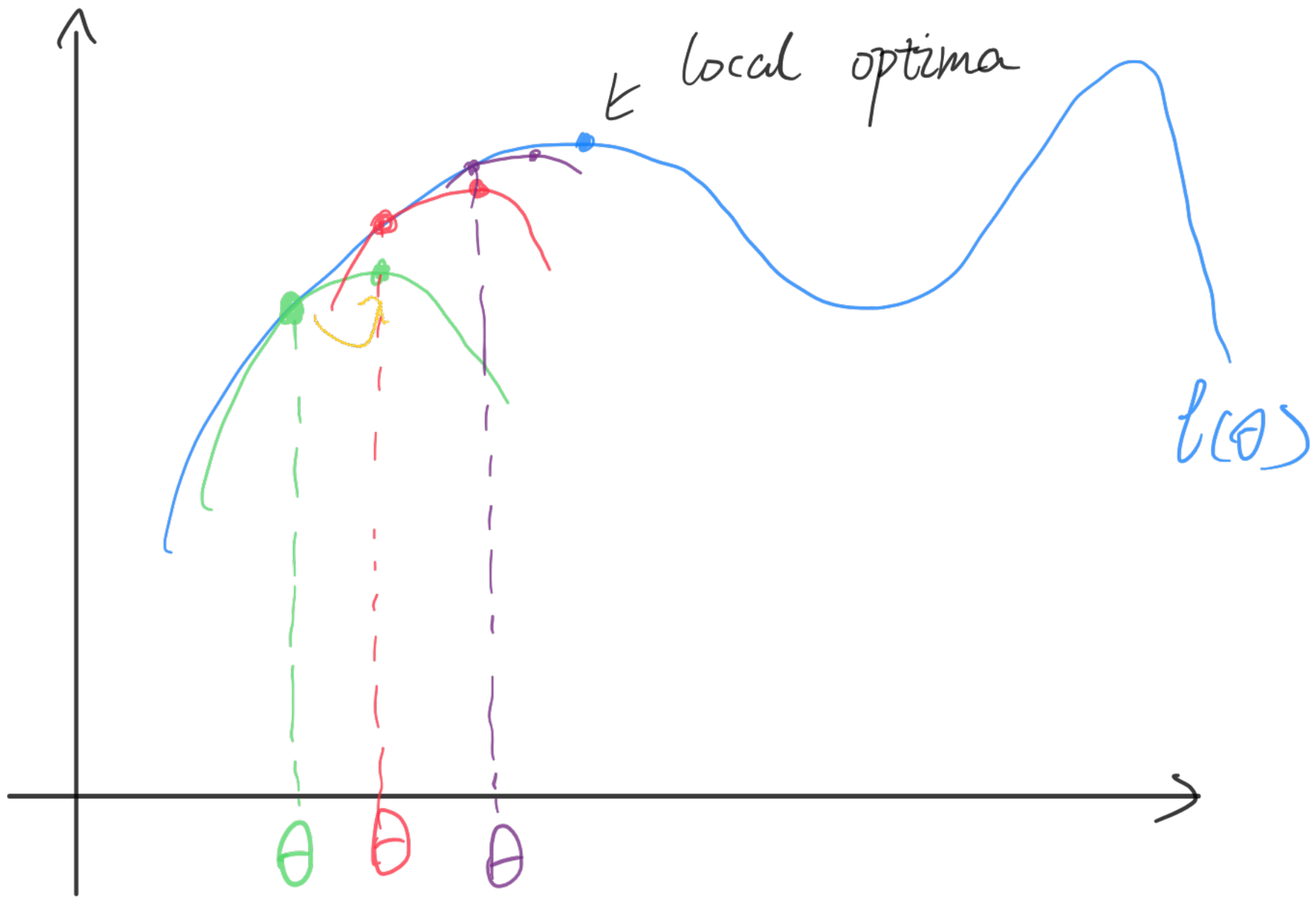
Let f be a convex function (i.e. $f''(x) > 0$), let x be a random variable, then $f(EX) \leq E[f(x)]$

Addendum

- If $f''(x) > 0$ (i.e. f is strictly convex), then $f(EX) = E[f(x)] \Longleftrightarrow X = E[X]$ with probablity 1 $(i.e. x$ is a constant$)$
- *Jensen's Equality in concave form:* Let f be a ***concave*** function (i.e. $f''(x) < 0$), let x be a random variable, then $f(EX) \geq E[f(x)]$



- Density Estimation



EM algorithm visualization:

- At E-step, constructing a lower bound (green curve) for the log-likelihood.
  Green curve has two properties
  1. Green curve is a lower bound (i.e. green curve lies blow the blue curve)
  2. Its value is equal to the blue curve at the current value of θ. (*This property guarantees that when you optimize the green function, you can improving blue function too.*)

     $$\log E_{z^{(i)}\sim Q_i}\left[\frac{P(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}\right] = E_{z^{(i)}\sim Q_i}\left[\log\frac{P(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}\right.$$

     To ensure this property, we need

     $$Q_i(z^{(i)}) = P(z^{(i)} \mid x^{(i)};\theta)$$
- At M-step, take the green curve and find its maximum
  Move θ from green value to the red value.

Lecture 15 Factor Analysis

Outline :

- **EM convergence**
  How to monitor if EM is converging
- **Gaussian Properties**
  Map the EM equations back to mixture of Gaussian models $Q_i(z^{(i)}) \Rightarrow w_j^{(i)}$
- **Factor Analysis** ⭐
  A useful model for datasets which is very high-dimensional but very few training examples
- **Gaussian Marginals & Conditionals**
- **EM Steps**
  Derive for the Factor Analysis model

Factor Analysis

107

Factor Analysis $z^{(i)} \sim \mathcal{N}$

Factor analysis can take very high dimensional data and model them to a lower dimensional subspace with a little bit of fuzz.

If the data doesn't lie in a subspace, model may not be the best model. But it's still a reasonable way to fit a Factor Analysis to n>>m dataset.

[Time Label]

A continuous z EM model

- **One Other View of EM**
- Coordinate Ascent $J(\theta, Q)$
- Comparison between Mixture of Gaussian & Factor Analysis
  - When $m \gg n$, use MG

    img
  - else FA

If `#Training Examples` < the dimension of the data, usual MLE function of covariance Σ will be singular(non-invertible)

Marginals and Conditionals of Gaussians

109

image-20220425150407136

Marginal : calculate $P(x_1)$ and we have $x_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$

Conditional : calculate $P(x_1 \mid x_2)$ we have $x_1 \mid x_2 \sim \mathcal{N}(\mu_{1|2}, \Sigma_{1|2})$

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$$

$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

EM for Factor Analysis

112

EM Steps

1. Derive $P(x, z)$

$$\begin{pmatrix} z \\ x \end{pmatrix} \sim \mathcal{N}(\mu_{x,z}, \Sigma)$$
$$z \sim \mathcal{N}(0, 1)$$
$$x = \mu + \Lambda z + \epsilon$$

image-20220425153323755

Simplify above equations. Finally we have

$$\mu_{x,z} = \begin{pmatrix} \vec{0} \\ \mu \end{pmatrix}$$
$$\Sigma = \begin{pmatrix} I & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \Psi \end{pmatrix}$$

Putting everything together, we have

$$\begin{bmatrix} z \\ x \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \vec{0} \\ \mu \end{bmatrix}, \begin{bmatrix} I & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \Psi \end{bmatrix}\right)$$

There's no parameters' closed form when solve derivatives of $P(x^{(i)})$'s log likelihood.

So, we use EM to solve these parameters.

2. **EM for Factor Analysis**
   - **E-Step**

$$Q_i(z^{(i)}) = P(z^{(i)} \mid x^{(i)}; \theta)$$
$$z^{(i)} \mid x^{(i)} \sim \mathcal{N}(\mu_{z^{(i)}|x^{(i)}}, \Sigma_{z^{(i)}|x^{(i)}})$$

     Where

$$\mu_{z^{(i)}|x^{(i)}} = \vec{0} + \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}(x^{(i)} - \mu)$$
$$\Sigma_{z^{(i)}|x^{(i)}} = I - \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}\Lambda$$

   - **M-step**

$$\Lambda = \left(\sum_{i=1}^{m}\left(x^{(i)} - \mu\right)\mu_{z^{(i)}|x^{(i)}}^T\right)\left(\sum_{i=1}^{m}\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}}\right)^{-1}$$

$$\mu = \frac{1}{m}\sum_{i=1}^{m}x^{(i)}$$

$$\Psi = \frac{1}{m}\sum_{i=1}^{m}x^{(i)}x^{(i)T} - x^{(i)}\mu_{z^{(i)}|x^{(i)}}^T\Lambda^T - \Lambda\mu_{z^{(i)}|x^{(i)}}x^{(i)T} + \Lambda\left(\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}}\right)$$

Tips:

Simplify integrals by

$$\int_{z^{(i)}} Q_i(z^{(i)})z^{(i)}dz^{(i)} = E[z^{(i)}] = \mu_{z^{(i)}|x^{(i)}}$$

Lecture 16 Independent Components Analysis & Reinforced Learning

Outline :

- **Independent Components Analysis**
  - CDF (cumulative distribution functions) b
  - ICA model
- Reinforcement Learning
  - MDP (Markov decision processes)

Independent Components Analysis 122

Reinforcement Learning and Control 128

Markov Decision Processes 129

MDP with unknown state transition probabilities 134

1. Initialize π randomly.
2. Repeat {
   (a) Execute π in the MDP for some number of trials.
   (b) Using the accumulated experience in the MDP, update our estimates for P_{sa} (and R, if applicable).
   (c) Apply value iteration with the estimated state transition probabilities and rewards to get a new estimated value function V.
   (d) Update π to be the greedy policy with respect to V.
   }

- Exploration vs Exploitation problem

When you acting a MDP, how aggressively of how greedy should you be at just taking actions to maximize your rewards?

Using **epsilon greedy** to solve this problem.

```
1.Initialize π randomly
2.Repeat{
        (a)Execute EPSILON-GREEDY π in the MDP
        (b)Using the accumulated experience in
        mates for $P_{sa}$ (and R,if applicable
        (c)Apply value iteration with the estim
        ities and rewards to get a new estimate
        (d)Update π to be the greedy policy wit
        }
```

Questions

- Is ϵ in **epsilon greedy** have to be constant?
  No, it doesn't have to be. *Boltzmann exploration
- Can you get a reward for reaching states you've never seen before?
  Intrinsic reinforcement learning / Intrinsic Motivation.
- How many actions should you take before updating π?
  Run as frequently as it can.

- State-Action Rewards

Rewards is a function mapping from states and actions to the rewards (i.e. $R(s,a) : S \times A \longmapsto \mathbb{R}$)

In this case, Bellman's equation is

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P_{sa}(s') V^*(s') \right)$$

Can use value iteration to solve $V^*(s)$, then you can get optimal policy, which is

$$\pi^*(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s'} P_{sa}(s') V^*(s') \right)$$

- Finite Horizon MDP

$\left(\mathcal{S}, \mathcal{A}, P_{sa}^{(t)}, T, R^{(t)}\right)$

Replace discount factor γ with a horizon time T, MDP will run a finite number of T steps.

- Action you take might depend on what time it is on the clock.
  Thus π should be time dependent (i.e. $\pi_t^*(s)$) Non-stationary policy
  Non-stationary state transitions $s_{t+1} \sim P_{s_t a_t}^{(t)}$
  Non-stationary Reward $R^{(t)}(s,a)$
  Examples
  - Changing dynamics
  - Weather forecasts
  - Industrial automation

How to solve for a finite horizon MDP

1. Define the optimal value function
   $$V_t(s) = \mathbb{E}\left[R^{(t)}(s_t, a_t) + \cdots + R^{(T)}(s_T, a_T) \mid s_t = s, \pi\right]$$
   $V_t(s)$ it the total payoff start on state s at time t execute π
2. Value iteration (a dynamic programming problem in this case)
   $\forall t < T, s \in \mathcal{S}: \quad V_t^*(s) := \max_{a \in \mathcal{A}} \left[R^{(t)}(s,a) + \mathbb{E}_{s' \sim P_{sa}^{(t)}}\left[V_{t+1}^*(s')\right]\right]$

   Base case (the final step)
   $$V_T^*(s) = \max_a R(s,a)$$

   Get the optimal policy
   $$\pi_t^*(s) = \arg\max_a \left(R(s,a) + \sum_{s'} P_{sa}(s') V_{t+1}^*(s') \right)$$

   Make R, P_{sa} be $R^{(t)}, P_{sa}^{(t)}$ for non-stationary problems
   - If make T infinite, value function $V_t^*(s)$ will be unbounded, thus simply make T infinite wouldn't get a discounted MDP formalism (traditional MDP) value iteration, we also need discount γ to ensure a value function bound.

Linear Quadratic Regulation (LQR)

Convenient to develop with the finite horizon setting $\left(\mathcal{S}, \mathcal{A}, P_{sa}^{(t)}, T, R^{(t)}\right)$, but also works with discounted MDP formalism $(\mathcal{S}, \mathcal{A}, P_{sa}, \gamma, R)$

Where to get A, B ?

- Learn from data
  Linear regression on m examples
- Linearize a non-linear model

A remarkable property

- Using LQR make the value function a quadratic function, can solve $V^\star$ exactly.

Dynamic programming for LQR

1. Base case

$$
\begin{aligned}
V_T^\star(s_T) &= \max_{a_T} R\left(s_T, a_T\right) \\
&= -s_T^T U s_T \\
\pi_T^\star(s_T) &= \vec{0}
\end{aligned}
$$

   Initialize $\Phi_T = -U, \Psi_T = \vec{0}$

2. The key step

   It's can be show that

$$
\begin{aligned}
&\text{if } V_{t+1}^*\left(s_{t+1}\right) = s_{t+1}^\top \Phi_{t+1} s_{t+1} + \Psi_{t+1} \\
&\text{then } V_t^*\left(s_t\right) = s_t^\top \Phi_t s_t + \Psi_t
\end{aligned}
$$

   Then we can solve LQR recursively.

   Recursive calculate Φ_t, Ψ_t using Φ_{t+1}, Ψ_{t+1} for $t = T-1, T-2, \ldots, 0$

$$
\begin{aligned}
\Phi_t &= A_t^\top \left(\Phi_{t+1} - \Phi_{t+1} B_t \left(B_t^\top \Phi_{t+1} B_t - W_t\right)^{-1} B_t \Phi_{t+1}\right) A_t - U_t \\
\Psi_t &= -\operatorname{tr}\left(\Sigma_t \Phi_{t+1}\right) + \Psi_{t+1}
\end{aligned}
$$

   Calculate L_t

   $\pi^\star(s_t) = L_t s_t$

$$
\begin{aligned}
L_t &= \left[\left(B_t^\top \Phi_{t+1} B_t - W_t\right)^{-1} B_t \Phi_{t+1} A_t\right] \\
\pi^\star(s_t) = a_t^* &= \left[\left(B_t^\top \Phi_{t+1} B_t - V_t\right)^{-1} B_t \Phi_{t+1} A_t\right] \cdot s_t \\
&= L_t \cdot s_t
\end{aligned}
$$

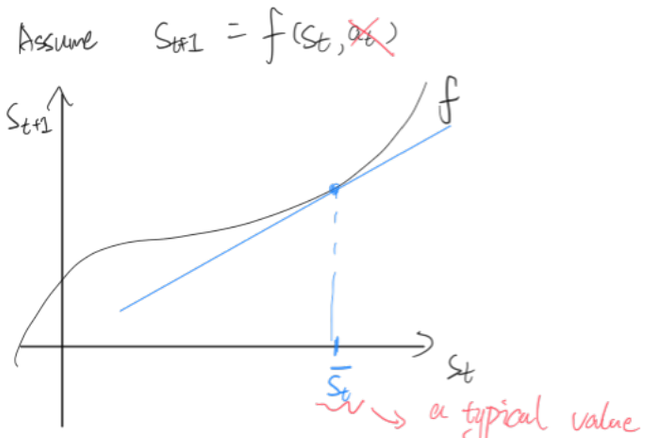
Takeaway: Optimal action is a linear function of state s_t

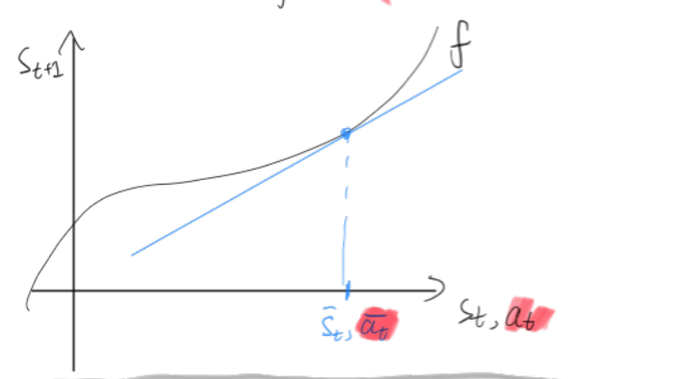
Fun Fact about LQR

- L_t depends on Φ_{t+1} but not Ψ_{t+1}, means that in order to take action, constant item for the quadratic function doesn't matter. ⚠️ So it's NOT NECESSARY to calculate Ψ in the LQR algorithm

  Thus $\pi^\star$, L_t don't depend on Σ_w, but $V^\star$ does.

- Linearize a non-linear model





$S_{t+1} = A S_t + B a_t$

Need to redefine S_t to add an intercept term

e.g. $S_t = \begin{pmatrix} 1 \\ x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{pmatrix}$

Lecture 17 MDPs & Value/Ploicy Iteration

Finite-State MDPs

131

Value Iteration

131

Focus on finding V^*

For absorbing state, set its P_{sa} to 0

Value iteration converges very quickly. (Due to γ, converges exponentially quickly)

1. For each state s, initialize $V(s) := 0$.
2. Repeat until convergence {

   For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$.

   }

repeatedly trying to update the estimated value function using Bellman Equations

- Synchronous Update

132

first compute the new values for V (s) for every state s, and then overwrite all the old values with the new values

- Asynchronous Updates 132

  loop over the states (in some order), updating the values one at a time

- Policy Iteration 132

  Focus on finding π^*

> 1. Initialize π randomly.
> 2. Repeat until convergence {
>
> (a) Let $V := V^\pi$.
>
> (b) For each state s, let $\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s') V(s')$.
>
> }

inner-loop repeatedly computes the value function for the current policy, and then updates the policy using the current value function

Lecture 18 Continuous State MDP & Model Simulation

Outline

- Discretization
- Models/Simulation
- Fitted Value Iteration

  Fitted Value Iteration works best with a model/simulator of the MDP

Continuous State MDPs 134

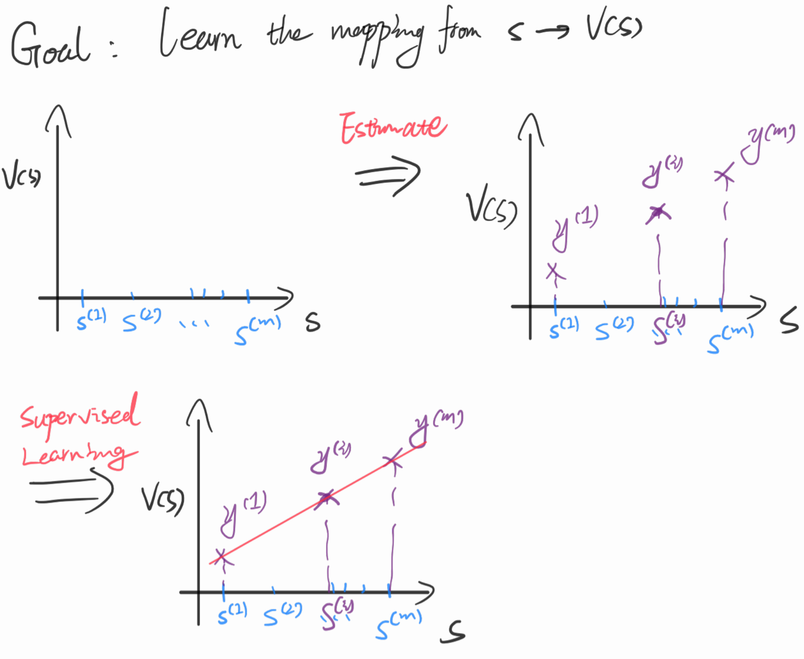
- Discretization 135

  discretization usually works extremely well for 1d and 2d problems (and has the advantage of being simple and quick to implement).

Value Function Approximation 137

- Fitted Value Iteration 140

> 1. Randomly sample m states $s^{(1)}, s^{(2)}, \ldots s^{(m)} \in S$.
> 2. Initialize $\theta := 0$.
> 3. Repeat {
>
> For $i = 1, \ldots, m$ {
>
> For each action $a \in A$ {
>
> Sample $s'_1, \ldots, s'_k \sim P_{s^{(i)}a}$ (using a model of the MDP).
>
> Set $q(a) = \frac{1}{k} \sum_{j=1}^{k} R(s^{(i)}) + \gamma V(s'_j)$
>
> // Hence, $q(a)$ is an estimate of $R(s^{(i)}) + \gamma \mathrm{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$.
>
> }
>
> Set $y^{(i)} = \max_a q(a)$.
>
> // Hence, $y^{(i)}$ is an estimate of $R(s^{(i)}) + \gamma \max_a \mathrm{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$.
>
> }
>
> // In the original value iteration algorithm (over discrete states)
>
> // we updated the value function according to $V(s^{(i)}) := y^{(i)}$.
>
> // In this algorithm, we want $V(s^{(i)}) \approx y^{(i)}$, which we'll achieve
>
> // using supervised learning (linear regression).
>
> Set $\theta := \arg\min_\theta \frac{1}{2} \sum_{i=1}^{m} \left(\theta^T \phi(s^{(i)}) - y^{(i)}\right)^2$
>
> }



- Lecture 19 Reward Model & Linear Dynamical System

  Outline

  - State-action rewards
  - Finite horizon MDP
  - Linear dynamical systems

    Can compute the exact value function (without approximation) even though the state space is continuous.

    - Model
    - LQR (Linear Quadratic Regulation)

- Lecture 20 RL Debugging and Diagnostics

  Outline :

  - RL debugging/diagnostics
  - Policy search
  - Conclusion

- RL debugging/diagnostics

  - Improving simulator
  - Modify value function
  - Modify RL algorithms

- (Direct) Policy Search

  Solve $\pi^\star$ directly (instead of using $V^\star$ to solve $\pi^\star$)
  DPS focus on coming up with the class of policies you'll entertain or come up with the set of functions you use to approximate the policy.
  New definition: A stochastic policy is a function $\pi : S \times A \longmapsto \mathbb{R}$ when $\pi(s, a)$ is the probability of taking action a in state s ($\sum_a \pi(s, a) = 1$)
  Goal : Find θ so that when we execute $\pi\theta(s, a)$, we maximize total payoff (expected sum of rewards), i.e.

  $$\max_\theta E\left[R\left(s_0, a_0\right) + ... + R\left(s_T, a_T\right) \mid \pi_\theta\right]$$

  How to do that Derive a stochastic gradient ascent algorithm as a function of θ solve the equation above.
  Reinforce Algorithm (*Very inefficient*)

  ```
  loop{
  /*Sample*/
  s_0, a_0, s_1,a_1 ... s_T,a_T
  /* Compute payoff*/
  $R(s_0,a_0),...,R(s_T,a_T)$
  /*Update $\theta$ (using gradient ascent)*/
  Implement $\theta$ update rule
  }
  ```

  θ update rule

  $$\theta := \theta + \alpha \left[\frac{\nabla \pi\theta(s_0, a_0)}{\pi\theta(s_0, a_0)} + \frac{\nabla \pi\theta(s_1, a_1)}{\pi\theta(s_1, a_1)} + ... + \frac{\nabla \pi\theta(s_T, a_T)}{\pi\theta(s_T, a_T)}\right] (R(s_0, a_0), ..., R(s_T, a_T)$$

  Note: One difference between policy search and estimated value function is that in Direct policy search s_0 is a fixed initial state s_0 or there's a fixed distribution over initial state s_0.
  Direct policy search also works for continuous value function, in that situation, we have $a = \theta^T +$ Gaussian noise.

- Direct Policy Search vs Value-function-based Approach

When to use DPS:

1. **POMDP (Partially Observable MDP)**
   At each step, get a partial (and potentially noisy) measurement of the state. Have to choose an action a using that.
   If we just have partially observed value of the state, even if we know $V^\star(s), \pi^\star(s)$, because we can't ensure what the state is, we still can not apply then. So we can only use DPS
   Can apply Kalman filter to estimate full state vector using partial observation state vectors, and plug them as features into policy search.
2. **When $\pi^\star$ is simpler than $V^\star$**
   For low-level control task, we often have simple map form $\mathcal{S} \longmapsto \mathcal{A}$ i.e. simpler policy $\pi^\star$. For multi-step reasoning problems, we should prefer to choose value function based approaches.

CONS for Reinforce Algorithm

- **Very inefficient**
  Gradient estimate for reinforcement algorithm turns out to be very noisy, even though the expected value is right.

(Direct) Policy Search

Value Function Approximation

- Newton's Method vs Gradient Ascent

- Newton's method converges quicker than Gradient ascent.
- While in high-dimensional (e.g. 1000, 1000000) problems, each iteration step costs more in Newton's Method.

gradient ascent

Newton's method