Fundamentals of Probability and Statistical Machine Learning for AI

INF8225 - Lesson 1b

See the following selection of slides from:

Data Mining

Practical Machine Learning Tools and Techniques

Slides from Chapter 9 of *Data Mining*
by I. H. Witten, E. Frank, M. A. Hall and C.J. Pal

Random variables

- In probabilistic approaches to machine learning it is common to think of data as observations arising from an underlying probability model for *random variables*
- Given a discrete random variable A, $P(A)$ is a function that encodes the probabilities for each of the categories, classes or states that A may be in
- For a continuous random variable x, $p(x)$ is a function that assigns a probability density to all possible values of x
- In contrast, $P(A=a)$ is the single probability of observing the specific event $A=a$

Notation

- The $P(A=a)$ notation is often simplified to simply $P(a)$, but one must remember if a was defined as a random variable or as an observation
- Similarly for the observation that continuous random variable x has the value x_1 it is common to write this as $p(x_1)=\mathrm{p}(x=x_1)$, a simplification of the longer but clearer notation

The product rule

- The *product rule*, sometimes referred to as the "fundamental rule of probability," states that the joint probability of random variables *A* and *B* can be written

$$P(A,B) = P(A \mid B)P(B)$$

- The product rule also applies when *A* and *B* are groups or subsets of events or random variables.

The sum rule

- The *sum rule* states that given the joint probability of variables X_1, X_2, …, X_N, the *marginal probability* for a given variable can be obtained by summing (or integrating) over all the other variables.
- For example, to obtain the marginal probability of X_1, sum over all the states of all the other variables:

$$P(X_1) = \sum_{x_2} \dots \sum_{x_N} P(X_1, X_2 = x_2, \dots, X_N = x_N)$$

Marginalization

- The previous notation can be simplified to

$$p(x_1) = \sum_{x_2} \dots \sum_{x_N} P(x_1, x_2, \dots, x_N)$$

- The sum rule generalizes to continuous random variables, ex. for $x_1, x_2, \dots, x_N$ we have

$$p(x_1) = \int_{x_2} \dots \int_{x_N} p(x_1, x_2, \dots, x_N) dx_2 \dots dx_N$$

- These procedures are known as *marginalization*
- They give us *marginal distributions* of the variables not included in the sums or integrals

Bayes' Rule

- Can be obtained by swapping *A* and *B* in the product rule and observing $P(B|A)P(A)=P(A|B)P(B)$ and therefore

$$P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A)}$$

- Suppose we have models for $P(A|B)$ and $P(B)$
  - We observe that $A=a$, and
  - we want to compute $P(B|A=a)$
  - $P(A=a|B)$ is referred to as the *likelihood*
  - $P(B)$ is the *prior* distribution of *B*
  - $P(B|A=a)$ is posterior distribution, obtained from:

$$P(A = a) = \sum_b P(A = a, B = b) = \sum_b P(A = a \mid B = b)P(B = b)$$

Maximum Likelihood

- Our goal is to estimate a set of parameters θ of a probabilistic model, given a set of *observations* $x_1, x_2, \ldots, x_n$.
- Maximum likelihood techniques assume that:
  1) the examples have no dependence on one another, the occurrence of one has no effect on the others, and
  2) each can be modeled in exactly the same way.
- These assumptions are often summarized by saying that events are *independent and identically distributed* (i.i.d.).

Maximum Likelihood

- The i.i.d. assumption corresponds to the use of a joint probability density function for all observations consisting of the product of the same probability model $p(x_i;\theta)$ applied to each observation independently.
- For *n* observations, this could be written as

$$p(x_1, x_2, \ldots, x_n;\theta) = p(x_1;\theta)p(x_2;\theta)\ldots p(x_n;\theta)$$

where each function $p(x_i;\theta)$ has the same θ

Maximum Likelihood

- The likelihood of our data can be written

$$\mathrm{L}(\theta; x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i; \theta)$$

- The data is fixed, but we can adjust θ so as to *maximize the likelihood* or *log-likelihood*

$$\theta_{ML} = \arg\max_{\theta} \sum_{i=1}^{n} \log p(x_i; \theta)$$

- We use the the log-likelihood as it is more numerically stable

Maximum a posteriori (MAP) parameter estimation

- If we treat our parameters as random variables we can compute the posterior

$$p(\theta \mid x_1, x_2, \ldots, x_n) = \frac{p(x_1, x_2, \ldots, x_n \mid \theta) p(\theta)}{p(x_1, x_2, \ldots, x_n)}$$

- We have used | or the "given" notation in place of ; to emphasize that θ is random, but
- Conditioned on a point estimate for the posterior we have a conditionally i.i.d. model
- MAP parameter estimation seeks

$$\theta_{MAP} = \arg\max_{\theta} \left[\sum_{i=1}^{n} \log p(x_i; \theta) + p(\theta) \right]$$

The chain rule of probability

- Results from applying the product rule recursively between a single variable and the rest of the variables
- The *chain rule* states that the joint probability of *n* attributes $A_{i=1...m}$ can be decomposed into the following product:

$$P(A_1, A_2, ..., A_n) = P(A_1) \prod_{i=1}^{n-1} P(A_{i+1} | A_i, A_{i-1}, .., A_1)$$

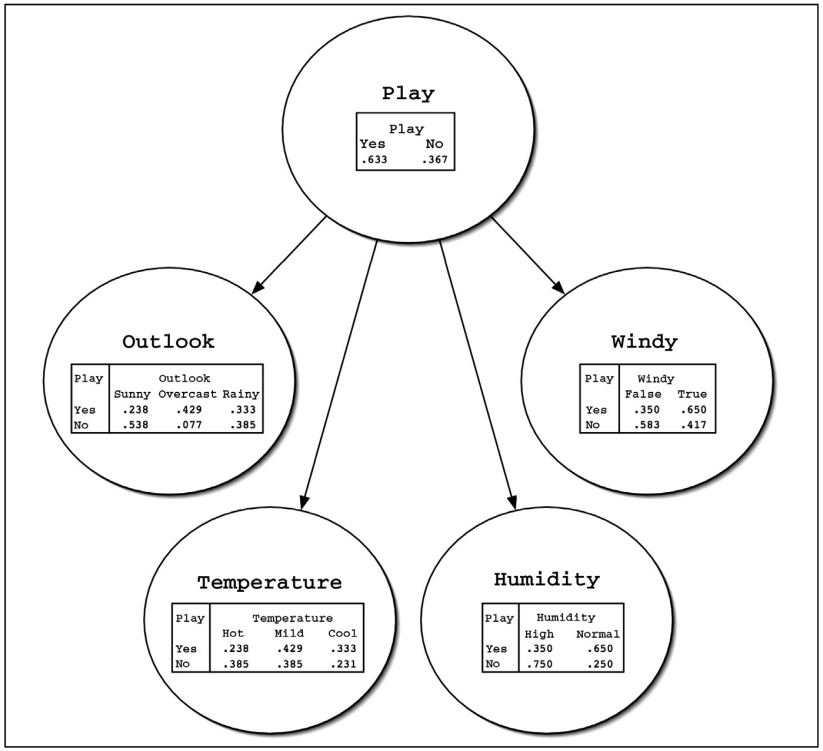
Bayesian networks

- The chain rule holds for any order for the A_is
- A Bayesian network is an acyclic graph,
- Therefore its nodes can be given an ordering where ancestors of node A_i have indices < i
- Thus a Bayesian network can be written

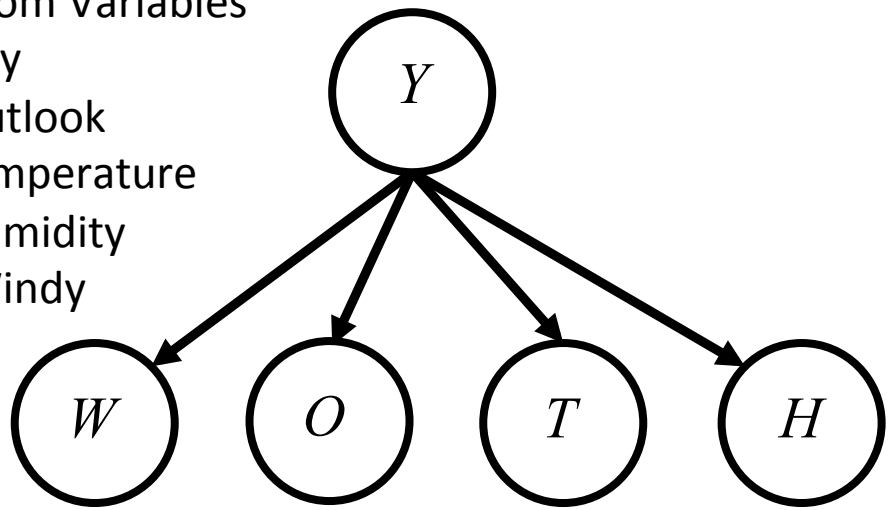
$$P(A_1, A_2, \ldots, A_n) = \prod_{i=1}^{n} P(A_i \mid \text{Parents}(A_i))$$

- When a variable has no parents, we use the unconditional probability of that variable

Example: Bayesian network model #1 for weather data



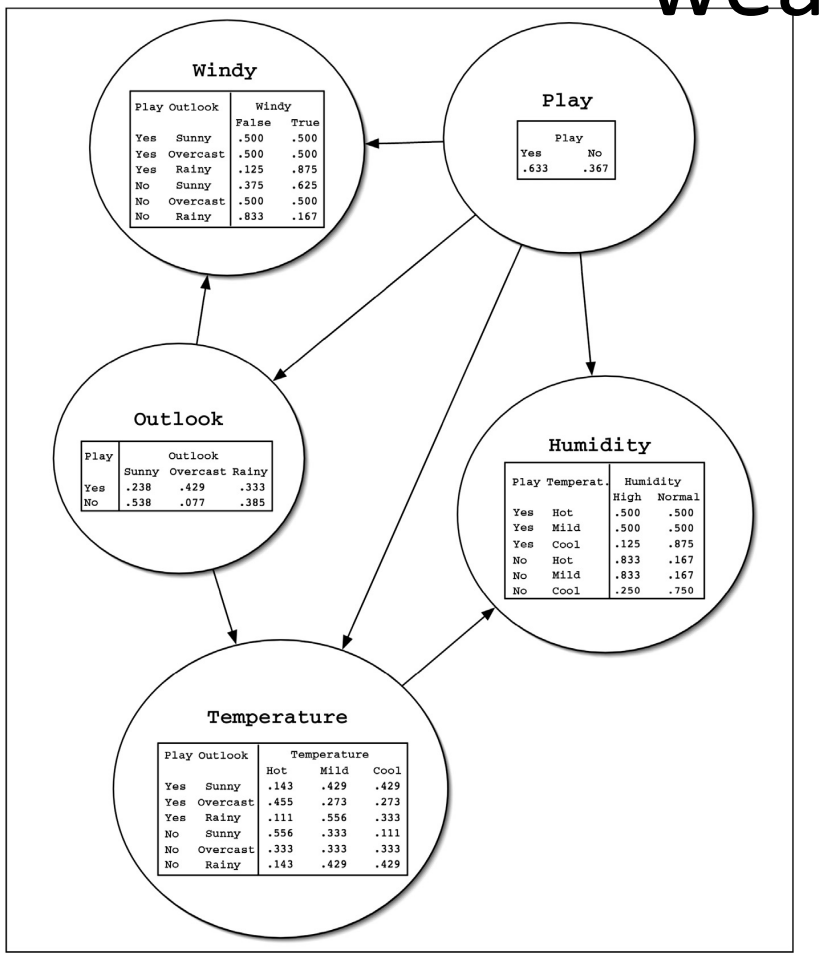
Random Variables
Y: Play
O: Outlook
T: Temperature
H: Humidity
W: Windy



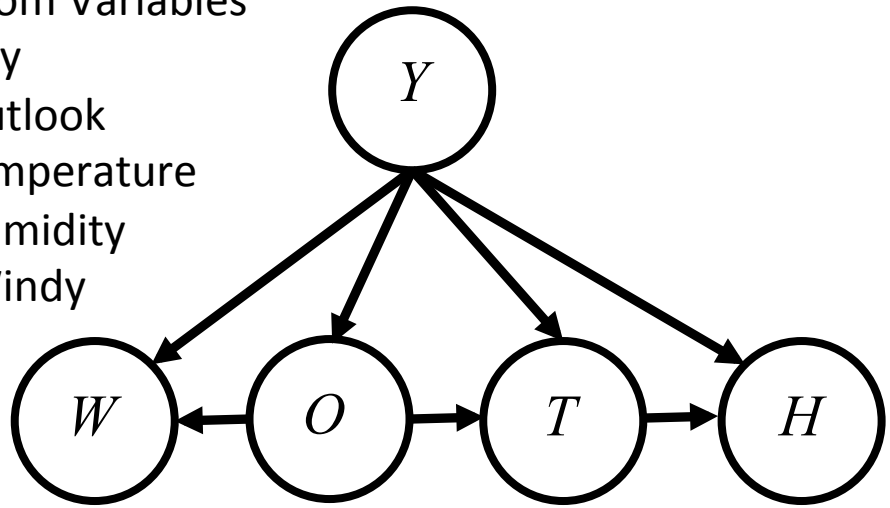
The graphs express the factorization below:

$$P(Y,O,.T,H,W) = P(W \mid Y)P(O \mid Y)P(T \mid Y)P(H \mid Y)P(Y)$$

Example Bayesian network #2 for the weather data



Random Variables
Y: Play
O: Outlook
T: Temperature
H: Humidity
W: Windy



The graphs express the factorization below:

$$P(Y,O,.T,H,W) = P(W \mid O,Y)P(O \mid Y)P(T \mid O,Y)P(H \mid T,Y)P(Y)$$

Estimating Bayesian network parameters

- The log-likelihood of a Bayesian network with *V* variables and *N* examples of complete variable assignments to the network is

$$\sum_{i=1}^{N} \log P(\{\tilde{A}_1, \tilde{A}_2, ..., \tilde{A}_V\}_i) = \sum_{i-1}^{N} \sum_{v=1}^{V} \log P(\tilde{A}_{v,i} \left| \text{Parents}(\tilde{A}_{v,i}); \Theta_v \right.)$$

  where the parameters of each conditional or unconditional distribution are given by Θ_v

- We use the $\tilde{A}_{v,i}$ notation to indicate the *i*th observation of variable *v*

Estimating probabilities in Bayesian networks

- The estimation problem *decouples* into separate estimation problems for each conditional or unconditional probability
- Unconditional probabilities can be written as

$$P(A = a) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\tilde{A}_i = a)$$

where $\mathbf{1}(\tilde{A}_i = a)$ is an indicator function returning 1 when the i^{th} observed value for A_i=a and 0 otherwise

Estimating conditional distributions

- Estimating conditional distributions in Bayesian networks is equally easy and amounts to simply counting configurations and dividing, ex.

$$P(B = b \mid A = a) = \frac{P(B = b, A = a)}{P(A = a)} = \frac{\sum_{i=1}^{N} \mathbf{1}(\tilde{A}_i = a, \tilde{B}_i = b)}{\sum_{i=1}^{N} \mathbf{1}(\tilde{A}_i = a)}.$$

- Zero counts cause problems and this motivates the use of Bayesian priors

Review Appendix A of

Data Mining
Practical Machine Learning Tools and Techniques

by I. H. Witten, E. Frank, M. A. Hall and C.J. Pal

See Theoretical Foundations, Appendix A.2 – Fundamental Elements of Probabilistic Methods

in

Data Mining

Practical Machine Learning Tools and Techniques

Slides for Chapter 9 of *Data Mining*

by I. H. Witten, E. Frank, M. A. Hall and C.J. Pal

Factor Graphs and Probability Models

Factor graphs

- Represent functions by factoring them into the product of local functions, each of which acts on a subset of the full argument set

$$F(x_1,\ldots,x_n) = \prod_{j=1}^{S} f_j(X_j)$$

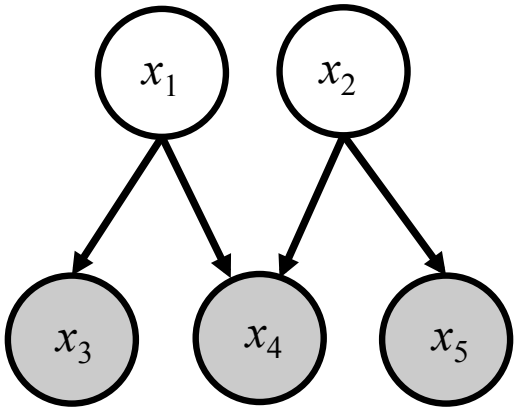
where X_j is a subset of the original set of arguments $\{x_1,\ldots,x_n\}$, $f_j(X_j)$ is a function of X_j, and j=1…S enumerates the argument subsets.

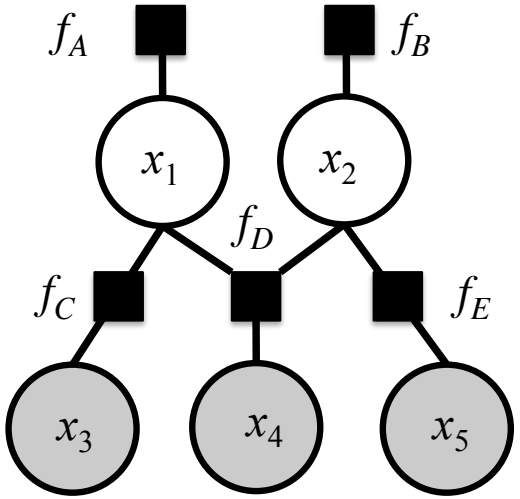
- A *factor graph* consists of variable nodes – circles – for each variable x_k and factor nodes – rectangles – for each function, with edges that connect each factor node to its variables.

Factor graph example

- A Bayesian network and its factor graph for

$$F(x_1,\ldots,x_5) = P(x_1)P(x_2)P(x_3 \mid x_1)P(x_4 \mid x_1,x_2)P(x_5 \mid x_2)$$
$$= f_A(x_1)f_B(x_2)f_C(x_3,x_1)f_D(x_4,x_1,x_2)f_E(x_5,x_2)$$





Computing Probabilities

Computing marginal probabilities

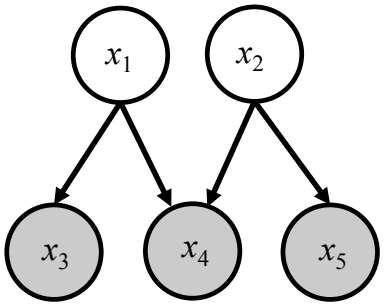
- The marginal for variable x_i is

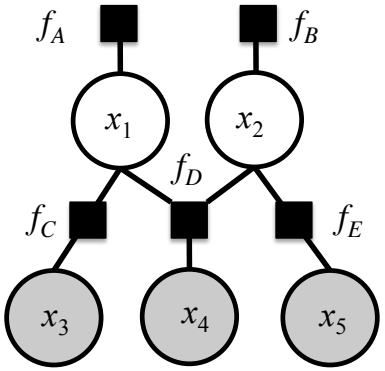
$$P(x_i) = \sum_{x_{j \neq i}} P(x_1, \ldots, x_n),$$

where the sum is over the states of all variables $x_j \neq x_i$

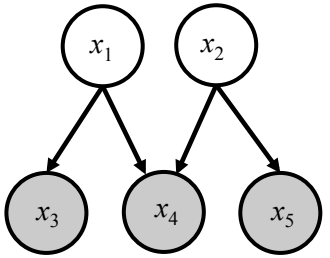
- Consider the task of computing the marginal *conditional* probability of variable x_3 given an observation for x_4 from a model where

$$P(x_1, \ldots, x_5) = P(x_1)P(x_2)$$
$$P(x_3 \mid x_1)P(x_4 \mid x_1, x_2)P(x_5 \mid x_2)$$





Marginal probabilities



- Since other variables in the graph have not been observed, they should be integrated out of the graphical model to obtain the desired result, ex.

$$P(x_3 \mid \tilde{x}_4) = \frac{P(x_3, \tilde{x}_4)}{P(\tilde{x}_4)} = \frac{P(x_3, \tilde{x}_4)}{\sum_{x_3} P(x_3, \tilde{x}_4)},$$ where the key quantity is

$$P(x_3, \tilde{x}_4) = \sum_{x_1} \sum_{x_2} \sum_{x_5} P(x_1, x_2, x_3, \tilde{x}_4, x_5)$$

$$= \sum_{x_1} \sum_{x_2} \sum_{x_5} P(x_1) P(x_2) P(x_3 \mid x_1) P(\tilde{x}_4 \mid x_1, x_2) P(x_5 \mid x_2).$$

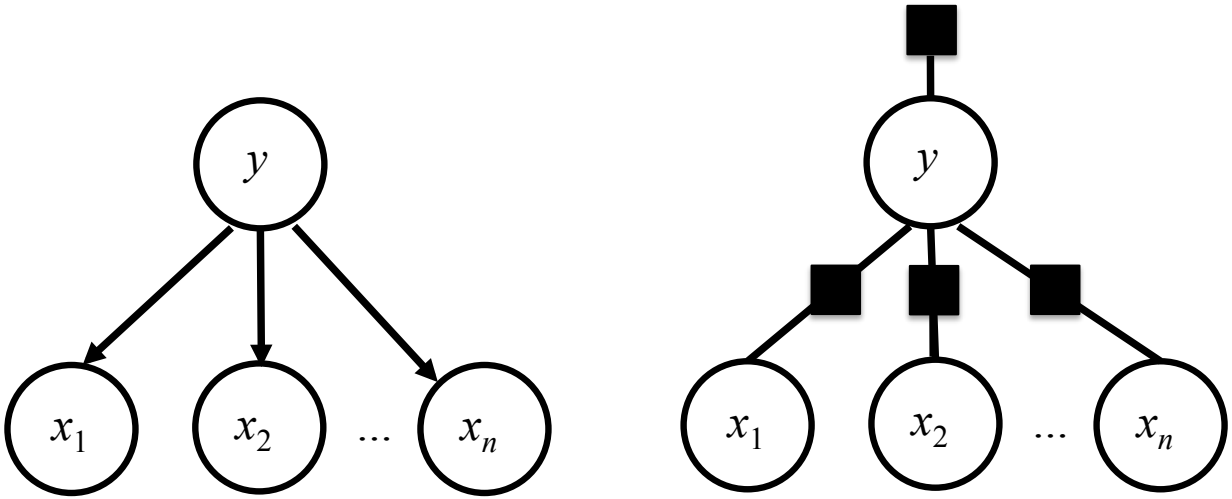
- However, this sum involves a large data structure containing the joint probability, composed of the products over the individual probabilities

Canonical Probabilistic and Statistical Models

Factor graphs for naïve Bayes models

- Naïve Bayes models have simple factor graphs

$$P(y, x_1, \ldots, x_n) = P(y) \prod_{i-1}^{n} P(x_i \mid y).$$

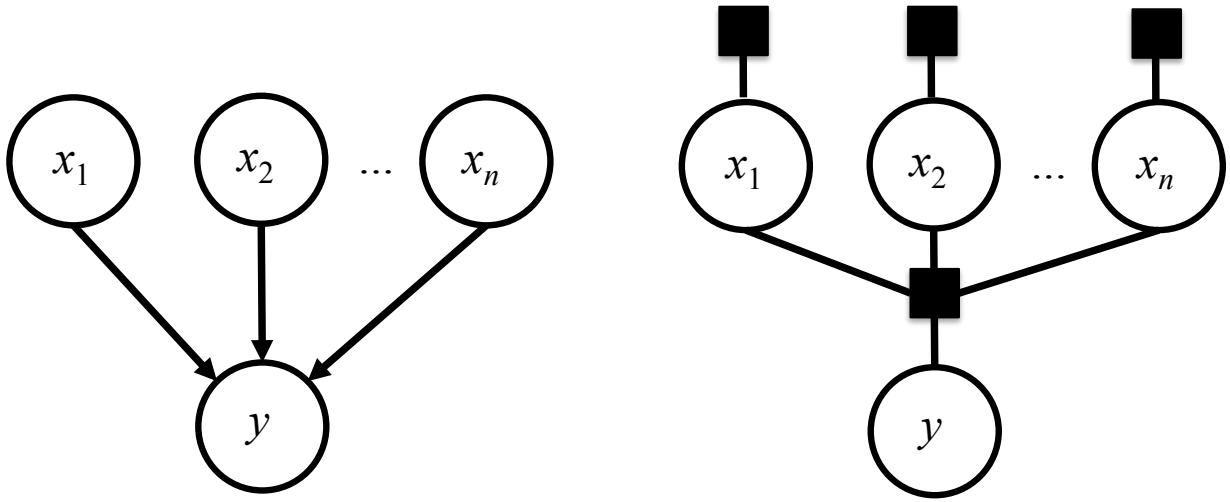


- Consider the number of parameters needed for each factor

An important observation

- Reversing the arrows in a naïve Bayes model leads to a variable that has many parents

$$P(y, x_1, \ldots, x_n) = P(y \mid x_1, \ldots, x_n) \prod_{i-1}^{n} P(x_i)$$



- Consider the # of parameters for p(y|x_1,…,x_n)

Logistic regression and factor graphs

- Rather than using a large table for the conditional distribution of a child y given many x_is, a logistic regression model can be used to reduce the number of parameters from exponential to linear
- Let's assume that all variables are binary
- Given a separate function $f_i(y,x_i)$ for each binary variable x_i, the conditional distribution defined by a logistic regression model can be expressed as shown in the next slide

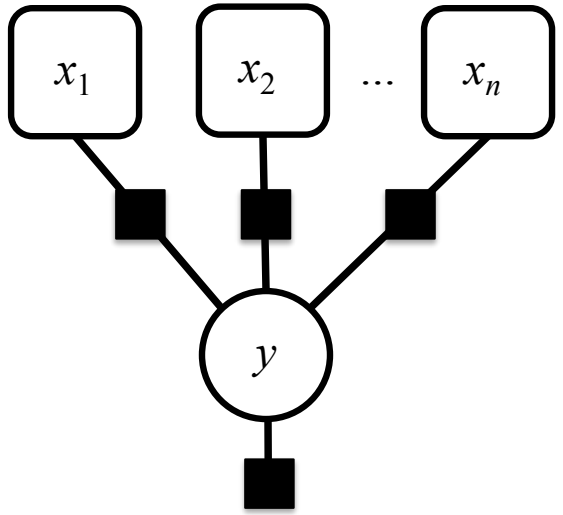
Logistic regression and factor graphs

- Logistic regression can be written as

$$P(y|x_1,\ldots,x_n)$$

$$= \frac{1}{Z(x_1,\ldots,x_n)} \exp\left(\sum_{i=1}^{n} w_i f_i(y,x_i)\right)$$

$$= \frac{1}{Z(x_1,\ldots,x_n)} \prod_{i=1}^{n} \phi_i(x_i,y).$$



where we note that the denominator *Z* is a *data dependent* a normalization term

- This factor graph resembles the Naïve Bayes model, but it is for a factorized *conditional* distribution
- Note: we have used rectangles for the x_i*s* because they are not being explicitly modeled as random variables

See Theoretical Foundations, Appendix A.1 - Matrix Algebra

in

Data Mining

Practical Machine Learning Tools and Techniques

Slides for Chapter 9 of *Data Mining*

by I. H. Witten, E. Frank, M. A. Hall and C.J. Pal

Multiclass Logistic Regression

(A single layer neural network)

Multiclass logistic regression

- A simple linear probabilistic classifier can be created using this parametric form

$$p(y \mid \mathbf{x}) = \frac{\exp\left(\sum_{k=1}^{K} w_k f_k(y,\mathbf{x})\right)}{\sum_{y} \exp\left(\sum_{k=1}^{K} w_k f_k(y,\mathbf{x})\right)} = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{k=1}^{K} w_k f_k(y,\mathbf{x})\right),$$

  where $y \in \{1,\ldots,N\}$, we use *K feature functions* $f_k(y,\mathbf{x})$ and K weights, w_k for the parameters
- We can perform learning using maximum conditional likelihood and N observations for both labels and features, $\{\tilde{y}_1,\ldots,\tilde{y}_N,\tilde{\mathbf{x}}_1,\ldots,\tilde{\mathbf{x}}_N\}$

Multiclass logistic regression and gradient computations

- The objective function (has no closed form solution)

$$L_{y|x} = \log \prod_{i=1}^{N} p(\tilde{y}_i \mid \tilde{\mathbf{x}}_i) = \sum_{i=1}^{N} \log p(\tilde{y}_i \mid \tilde{\mathbf{x}}_i)$$

- Soln: gradient descent - the derivative w.r.t. one weight is

$$\frac{\partial}{\partial w_j} \log(p(\tilde{y} \mid \tilde{\mathbf{x}})) = \frac{\partial}{\partial w_j} \left\{ \log \left(\frac{1}{Z(\tilde{\mathbf{x}})} \exp \left(\sum_{k=1}^{K} w_k f_k(\tilde{y}, \tilde{\mathbf{x}}) \right) \right) \right\}$$

$$= \frac{\partial}{\partial w_j} \left\{ \left[\sum_{k=1}^{K} \underbrace{w_k f_k(\tilde{y}, \tilde{\mathbf{x}})}_{\text{easy part}} \right] - \underbrace{\log Z(\tilde{\mathbf{x}})}_{\text{cool part}} \right\}$$

$$= f_{k=j}(\tilde{y}, \tilde{\mathbf{x}}) - \frac{\partial}{\partial w_j} \left\{ \log \left[\sum_{y} \exp \left(\sum_{k=1}^{K} w_k f_k(y, \tilde{\mathbf{x}}) \right) \right] \right\},$$

we see the derivative breaks apart into two key terms

- The first term, the "easy part" involves terms that disappear when $w_k \neq, w_j$ leaving only the term shown on the left

Understanding gradients for logistic regression

- The second term, while seemingly daunting, has a "cool" derivative that yields an intuitive and interpretable result:

$$-\frac{\partial}{\partial w_j}\left\{\log Z(\tilde{\mathbf{x}})\right\} = -\frac{\partial}{\partial w_j}\left\{\log\left[\sum_y \exp\left(\sum_{k=1}^{K} w_k f_k(y,\tilde{\mathbf{x}})\right)\right]\right\}$$

$$= -\frac{\sum_y \exp\left(\sum_{k=1}^{K} w_k f_k(y,\tilde{\mathbf{x}})\right) f_j(y,\tilde{\mathbf{x}})}{\sum_y \exp\left(\sum_{k=1}^{K} w_k f_k(y,\tilde{\mathbf{x}})\right)} = -\sum_y p(y|\tilde{\mathbf{x}}) f_j(y,\tilde{\mathbf{x}}) = -\mathrm{E}[f_j(y,\tilde{\mathbf{x}})]_{p(y|\tilde{\mathbf{x}})},$$

  which corresponds to the expectation of the feature function under the probability distribution given by the model with the current parameter settings

- Using vectorized **w** and feature functions **f** we have

$$\frac{\partial}{\partial \mathbf{w}} L_{y|x} = \sum_{i=1}^{N}\left[\mathbf{f}(\tilde{y}_i \mid \tilde{\mathbf{x}}_i) - \mathrm{E}[\mathbf{f}(y_i \mid \tilde{\mathbf{x}}_i)]_{P(y_i|\tilde{\mathbf{x}}_i)}\right]$$

Reformulating multiclass logistic regression

- Could alternatively formulate logistic regression as:

$$p(y = c \mid \mathbf{x}) = \frac{\exp\left(\mathbf{w}_c^T \mathbf{x}\right)}{\sum_{\mathbf{y}} \exp\left(\mathbf{w}_y^T \mathbf{x}\right)},$$

  where *y* is an integer index, the weights are encoded into a vector of length *K,* and
- The features **x** have been re-defined as the result of evaluating the feature functions $f_k(y,\mathbf{x})$ in such a way that there is no difference between the features given by $f_k(y=i,\mathbf{x})$ and $f_k(y=j,\mathbf{x})$.
- This form is widely used for the last layer in neural network models, where it is referred to as the *softmax* function

Matrix vector formulation of multiclass logistic regression

- The information concerning class labels could be encoded into a *multinomial* or *one-hot vector* **y**, which is all zeros except for a single 1 in the dimension that represents the correct class label—for example, $\mathbf{y} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \end{bmatrix}^T$ for the second class.
- The weights form a matrix, $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \dots & \mathbf{w}_K \end{bmatrix}^T$ and the biases form a vector, $\mathbf{b} = \begin{bmatrix} b_1 & b_2 & \dots & b_K \end{bmatrix}^T$
- The model can be formulated so as to yield *vectors* of probabilities

Logistic regression in matrix vector form

- With the previous definitions, we can now write

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{\exp\left(\mathbf{y}^T \mathbf{W}\mathbf{x} + \mathbf{y}^T \mathbf{b}\right)}{\sum_{\mathbf{y} \in Y} \exp\left(\mathbf{y}^T \mathbf{W}^T \mathbf{x} + \mathbf{y}^T \mathbf{b}\right)},$$

  where the denominator sums over each possible label,

$$\mathbf{y} \in Y, \quad Y = \{[\ 1\ \ 0\ \ 0\ \ \ldots\ \ 0\ ]^T, \ldots, [\ 0\ \ 0\ \ 0\ \ \ldots\ \ 1\ ]^T\}$$

- Re-defining **x** as $\mathbf{x} = [\mathbf{x}^T\ 1]^T$ and the parameters as a matrix of the form

$$\boldsymbol{\theta} = [\mathbf{W}\ \ \mathbf{b}] = \begin{bmatrix} \mathbf{w}_1^T & b_1 \\ \mathbf{w}_2^T & b_2 \\ \vdots & \vdots \\ \mathbf{w}_k^T & b_k \end{bmatrix},$$

Logistic regression in matrix vector form

- We can now write logistic regression very compactly as

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{\exp\left(\mathbf{y}^T \boldsymbol{\theta} \mathbf{x}\right)}{\sum_{\mathbf{y} \in Y} \exp\left(\mathbf{y}^T \boldsymbol{\theta} \mathbf{x}\right)} = \frac{1}{Z(\mathbf{x})} \exp\left(\mathbf{y}^T \boldsymbol{\theta} \mathbf{x}\right)$$

- The gradient of the log-conditional-likelihood with respect to the parameter *matrix* $\boldsymbol{\theta}$ is

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\theta}} \log \prod_i p(\tilde{\mathbf{y}}_i \mid \tilde{\mathbf{x}}_i; \boldsymbol{\theta}) &= \sum_{i=1}^{N} \left[\frac{\partial}{\partial \boldsymbol{\theta}} (\tilde{\mathbf{y}}_i^T \boldsymbol{\theta} \tilde{\mathbf{x}}_i) - \frac{\partial}{\partial \boldsymbol{\theta}} \log Z(\tilde{\mathbf{x}}_i) \right] \\
&= \sum_{i=1}^{N} \tilde{\mathbf{y}}_i \tilde{\mathbf{x}}_i^T - \sum_{i=1}^{N} \sum_{\mathbf{y} \in Y} P(\mathbf{y} \mid \tilde{\mathbf{x}}_i) \mathbf{y} \tilde{\mathbf{x}}_i^T \\
&= \sum_{i=1}^{N} \tilde{\mathbf{y}}_i \tilde{\mathbf{x}}_i^T - \sum_{i=1}^{N} \mathrm{E}[\mathbf{y} \tilde{\mathbf{x}}_i^T]_{P(\mathbf{y} \mid \tilde{\mathbf{x}}_i)}
\end{aligned}$$

- The second term transforms into a vector of probabilities for the classes of **y** under the model, multiplied by the observed **x** transpose

Gradient descent

- Given a conditional probability model $p(y \mid \mathbf{x}; \boldsymbol{\theta})$
- Parameter vector $\boldsymbol{\theta}$, data $\tilde{y}_i, \tilde{\mathbf{x}}_i, i = 1 \ldots N$
- Prior on parameters, $p(\boldsymbol{\theta}; \lambda)$ with hyperparameter, λ
- Gradient descent with learning rate η can be written as:

$$\boldsymbol{\theta} = \boldsymbol{\theta}_o \quad \text{// initialize parameters}$$

$$\text{while converged} \;==\; \text{FALSE}$$

$$\mathbf{g} \;=\; \frac{\partial}{\partial \boldsymbol{\theta}} \left[-\sum_{i=1}^{N} \log p(\tilde{y}_i \mid \tilde{\mathbf{x}}_i; \boldsymbol{\theta}) - \log p(\boldsymbol{\theta}; \lambda) \right]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \mathbf{g}$$

- For convex models the change in the loss or the parameters is often monitored and the algorithm is terminated when it stabilizes

Mini-batch based stochastic gradient descent (SGD)

- Stochastic gradient descent updates model parameters according to the gradient computed from one example
- The mini-batch variant uses a small subset of the data and bases updates to parameters on the average gradient over the examples in the batch
- This operates just like the regular procedure: initialize the parameters, enter a parameter update loop, and terminate by monitoring a validation set
- Normally these batches are randomly selected disjoint subsets of the training set, perhaps shuffled after each epoch, depending on the time required to do so

Mini-batch based SGD

- Each pass through a set of mini-batches that represent the complete training set is an *epoch*
- Using the empirical risk plus a regularization term as the objective function, updates are

$$\theta^{\text{new}} \leftarrow \theta - \eta_t \left[\frac{1}{B_k} \sum_{i \in \mathrm{I}} \left[\frac{\partial}{\partial \theta} L(f(\mathbf{x}_i;\theta), \mathbf{y}_i) \right] + \frac{B_k}{N} \lambda \frac{\partial}{\partial \theta} R(\theta) \right]$$

- η_t is the learning rate and may depend on the epoch t
- The batch is represented by a set of indices I=I(t,k) into the original data; the kth batch has B_k examples
- N is the size of the training set
- $L(f(\mathbf{x}_i;\theta),\mathbf{y}_i)$ is the loss for example $\mathbf{x}_i$, label $\mathbf{y}_i$, params θ
- R(θ) is the regularizer, with weight λ

Mini-batches

- Typically contain two to several hundred examples
  - For large models the choice may be constrained by resources
- Batch size often influences the stability and speed of learning; some sizes work particularly well for a given model and data set.
- Sometimes a search is performed over a set of potential batch sizes to find one that works well, before doing a lengthy optimization.
- The mix of class labels in the batches can influence the result
  - For unbalanced data there may be an advantage in pre-training the model using mini-batches in which the labels are balanced, then fine-tuning the upper layer or layers using the unbalanced label statistics.

Momentum

- As with regular gradient descent, 'momentum' can help the optimization escape plateaus in the loss
- Momentum is implemented by computing a moving average: $\Delta\theta = -\eta\nabla_\theta L(\theta) + \alpha\Delta\theta^{\text{old}}$
  - where the first term is the current gradient of the loss times a learning rate
  - the second term is the previous update weighted by $\alpha \in [0,1]$
- Since the mini- batch approach operates on a small subset of the data, this averaging can allow information from other recently seen mini-batches to contribute to the current parameter update
- A momentum value of 0.9 is often used as a starting point, but it is common to hand-tune it, the learning rate, and the schedule used to modify the learning rate during the training process

Learning rate schedules

- The learning rate is a critical choice when using mini-batch based stochastic gradient descent.
- Small values such as 0.001 often work well, but it is common to perform a logarithmically spaced search, say in the interval [10^{-8}, 1], followed by a finer grid or binary search.
- The learning rate may be adapted over epochs t to give a learning rate schedule, ex. $\eta_t = \eta_0 (1 + \varepsilon t)^{-1}$
- A fixed learning rate is often used in the first few epochs, followed by a decreasing schedule
- Many other options, ex. divide the rate by 10 when the validation error rate ceases to improve

Mini-batch SGD pseudocode

$\theta = \theta_o$ // initialize parameters

$\Delta\theta = 0$

$t = 0$

while converged == FALSE

$\{I_1, \ldots, I_K\} = \text{shuffle}(X)$ // create K mini-batches

for $k = 1 \ldots K$

$$\mathbf{g} = \frac{1}{B_k} \sum_{i \in I_k} \left[\frac{\partial}{\partial \theta} L(f(\mathbf{x}_i; \theta), \mathbf{y}_i) \right] + \frac{B_k}{N} \lambda \frac{\partial}{\partial \theta} R(\theta)$$

$$\Delta\theta \leftarrow -\eta_t \mathbf{g} + \alpha \Delta\theta$$

$$\theta \leftarrow \theta + \Delta\theta$$

end

$t = t + 1$

end

Second order methods

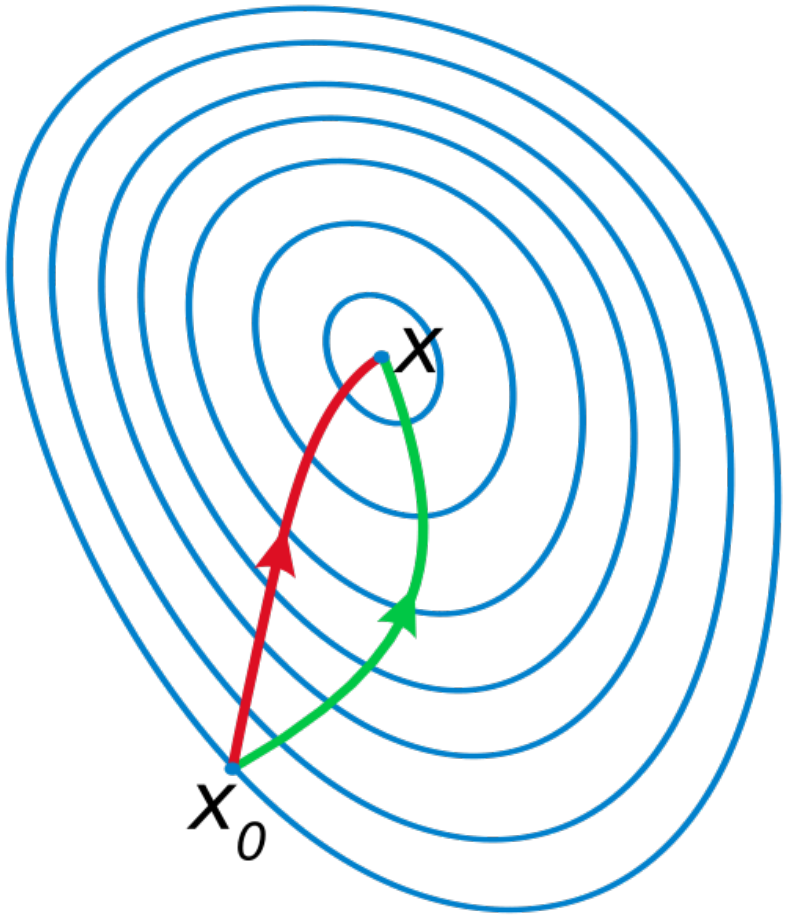
- Alternatively, gradient descent can be based on the second derivative by computing the Hessian matrix **H** at each iteration and using updates

$$\mathbf{H} = \frac{\partial^2}{\partial \boldsymbol{\theta}^2}\left[-\sum_{i=1}^{N} \log p(\tilde{y}_i \mid \tilde{\mathbf{x}}_i; \boldsymbol{\theta}) - \log p(\boldsymbol{\theta}; \boldsymbol{\lambda})\right],$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{H}^{-1}\mathbf{g}.$$

- To understand why this is possible and the relationship to learning rates, see Appendix A.1

Comparer la descente du gradient aux méthodes de deuxième ordre



- Vert : Descente du gradient

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \lambda \mathbf{g}$$

- Rouge : La méthode de Newton

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \lambda \mathbf{H}^{-1} \mathbf{g}$$

The Taylor Series and Second Order Optimization Methods

The Taylor expansion of a function near point x_o can be written

$$f(x) = f(x_o) + \frac{f'(x_o)}{1!}(x - x_o) + \frac{f''(x_o)}{2!}(x - x_o)^2 + \frac{f^{(3)}(x_o)}{3!}(x - x_o)^3 + \cdots$$

Using the approximation up to the second-order (squared) terms in x, taking the derivative, setting the result to zero, and solving for x, gives

$$\begin{aligned} 0 &= \frac{d}{dx}\left[f(x_o) + f'(x_o)(x - x_o) + \frac{f''(x_o)}{2}(x - x_o)^2 \right] \\ &= f'(x_o) + f''(x_o)(x - x_o), \text{ thus solving for } \delta x \equiv (x - x_o) \\ \Rightarrow \delta x &= -\frac{f'(x_o)}{f''(x_o)}, \quad \text{or} \quad x = x_o - \frac{f'(x_o)}{f''(x_o)}. \end{aligned}$$

These are the update equations in 1D. What about ND ?

N-D Second Order Methods

This generalizes to the vector version of a Taylor series for a scalar function with matrix arguments, where

$$f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}_o) + \mathbf{g}_o^T(\boldsymbol{\theta} - \boldsymbol{\theta}_o) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_o)^T \mathbf{H}_o(\boldsymbol{\theta} - \boldsymbol{\theta}_o) + \cdots,$$

$$\mathbf{g}_o = \frac{df}{d\boldsymbol{\theta}_o}, \mathbf{H}_o = \frac{d}{d\boldsymbol{\theta}_o}\frac{df}{d\boldsymbol{\theta}_o} = \frac{d^2 f}{d\boldsymbol{\theta}_o^2}.$$

Using the identities given in the previous section, taking the derivative with respect to the parameter vector $\boldsymbol{\theta}$, setting it to zero and then solving for the point at which the quadratic approximation to the function would be zero, yields

$$\frac{df(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = 0 = \mathbf{g}_o + \mathbf{H}_o(\boldsymbol{\theta} - \boldsymbol{\theta}_o), \Delta\boldsymbol{\theta} \equiv (\boldsymbol{\theta} - \boldsymbol{\theta}_o)$$

$$\Rightarrow \Delta\boldsymbol{\theta} = -\mathbf{H}_o^{-1}\mathbf{g}_o.$$

The Hessian…

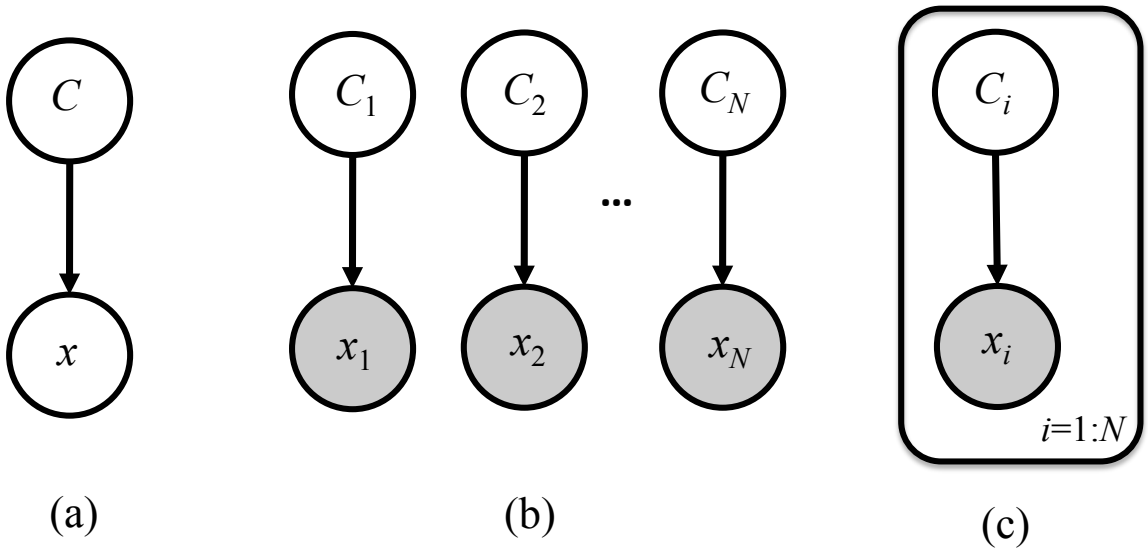
- Can be scary

- Which is why techniques like LBFGS just approximate the Hessian

Probability Models, Plate Notation and the Gaussian Mixture Model

Plate notation

- A "plate" is simply a box around a Bayesian network that denotes a certain number of replications of it, one for each data instance
- The plate below indicates i=1…N networks, each with an observed value for x_i and hidden variable C_i
- Plate notation captures a model for the joint probability of the entire data with a simple picture.



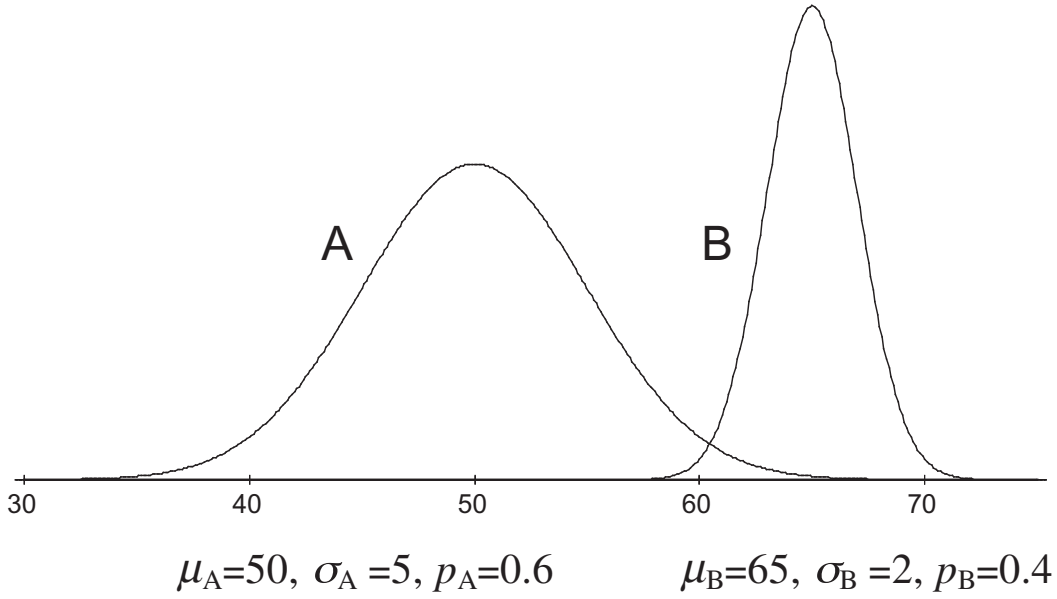
Clustering with a Gaussian Mixture

- Given the data on the left *without the labels A and B*, we wish to estimate a model for a two class Gaussian Mixture Model (GMM) on the right

Data

A 51	A 46	B 65
A 43	B 64	A 46
B 62	A 51	A 39
B 64	A 52	B 62
A 45	B 62	B 64
A 42	A 49	A 52
A 46	A 48	B 63
A 45	B 62	B 64
A 45	A 43	A 48
B 62	A 40	B 64
A 47	A 48	A 48
A 52	B 64	A 51
B 64	A 51	A 48
A 51	B 63	B 64
B 65	A 43	A 42
A 48	B 65	A 48
A 49	B 66	A 41

Model



Clustering and probability density estimation

- If we had data belonging to two known classes A and B, each having a normal distribution with means and standard deviations μ_A and σ_A for class A, and μ_B and σ_B for class B
- We could define a model whereby samples are taken from these distributions, using cluster A with probability p_A and cluster B with probability p_B (where $p_A + p_B = 1$),
- Sampling we might obtain the data in the next slide
- Now, imagine given the dataset without the classes—just the numbers—and being asked to determine the five parameters that characterize the model: μ_A, σ_A, μ_B, σ_B, and p_A (the parameter p_B can be calculated directly from p_A)

Estimating Gaussian parameters

- If we knew which of the two distributions each instance came from, finding the five parameters would be easy—just estimate the mean and standard deviation for $n=n_A$ or $n=n_B$ samples x_1, x_2, …, x_n for each cluster, A and B

$$\mu = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

$$\sigma^2 = \frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \ldots + (x_n - \mu)^2}{n-1}$$

- To estimate the fifth parameter p_A, just take the proportion of the instances that are in the A cluster, then $p_B=1-p_A$.

Motivating the EM algorithm

- If you knew the five parameters, finding the (posterior) probabilities that a given instance comes from each distribution would be easy
- Given an instance x_i, the probability that it belongs to cluster A is

$$P(\mathrm{A}|x_i) = \frac{P(x_i \mid \mathrm{A}) \cdot P(\mathrm{A})}{P(x_i)} = \frac{N(x_i;\mu_\mathrm{A},\sigma_\mathrm{A})p_\mathrm{A}}{N(x_i;\mu_\mathrm{A},\sigma_\mathrm{A})p_\mathrm{A} + N(x_i;\mu_\mathrm{B},\sigma_\mathrm{B})p_\mathrm{B}}$$

where $N()$ is the normal or Gaussian distribution

$$N(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The EM algorithm for a GMM

- Start with a random (but reasonable) assignment to the parameters
- Compute the posterior distribution for the cluster assignments for each example
- Update the parameters based on the expected class assignments - where probabilities act like weights.
- If w_i is the probability that instance *i* belongs to cluster A, the mean and std. dev. are

$$\mu_A = \frac{w_1 x_1 + w_2 x_2 + \ldots + w_n x_n}{w_1 + w_2 + \ldots + w_n}$$

$$\sigma_A^2 = \frac{w_1 (x_1 - \mu)^2 + w_2 (x_2 - \mu)^2 + \ldots + w_n (x_n - \mu)^2}{w_1 + w_2 + \ldots + w_n}$$

The EM algorithm

- Optimizes the marginal likelihood, obtained by marginalizing over the two components of the Gaussian mixture
- The marginal likelihood is a measure of the "goodness" of the clustering and it increases at each iteration of the EM algorithm.
- In practice we use the log marginal likelihood

$$\log \prod_{i=1}^{n} P(x_i) = \sum_{i=1}^{n} \log \sum_{c_i} P(x_i \mid c_i) \cdot P(c_i)$$

$$= \sum_{i=1}^{n} \log \left[N(x_i; \mu_{\mathrm{A}}, \sigma_{\mathrm{A}}) p_{\mathrm{A}} + N(x_i; \mu_{\mathrm{B}}, \sigma_{\mathrm{B}}) p_{\mathrm{B}} \right]$$

Extending the mixture Model

- The Gaussian distribution generalizes to n-dimensions
- Consider a two-dimensional model consisting of independent Gaussian distributions for each dimension
- We can transform from scalar to matrix notation for a two dimensional Gaussian distribution as follows:

$$P(x_1, x_2) = \frac{1}{\sqrt{2\pi}\sigma_1}\exp\left[-\frac{(x_1-\mu_1)^2}{2\sigma_1^2}\right]\frac{1}{\sqrt{2\pi}\sigma_2}\exp\left[-\frac{(x_2-\mu_2)^2}{2\sigma_2^2}\right]$$

$$= (2\pi)^{-1}\left(\sigma_1^2\sigma_2^2\right)^{-1/2}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\begin{bmatrix}\sigma_1^2 & 0\\ 0 & \sigma_2^2\end{bmatrix}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

$$= (2\pi)^{-1}|\boldsymbol{\Sigma}|^{-1/2}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\},$$

- $\boldsymbol{\Sigma}$ is the covariance *matrix,* $|\boldsymbol{\Sigma}|$ is its determinant, the vector $\mathbf{x} = [x_1\ x_2]^T$, and the mean *vector* $\boldsymbol{\mu} = [\mu_1\ \mu_2]^T$

The multivariate Gaussian distribution

- Can be written in the following general form

$$P(x_1, x_2, \ldots, x_d) = (2\pi)^{-d/2} |\boldsymbol{\Sigma}|^{-1/2} \exp\left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}.$$

- The equation for estimating the covariance matrix is

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T.$$

- The mean is simply

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i.$$

Classical Simple Statistical Models as Conditional Probability Models

Conditional continuous probability models

- Consider a model where the conditional probability for y_i given x_i is a Gaussian with mean given by a linear function of x:

$$p(y_i \mid x_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{\left\{ y_i - (\theta_0 + \theta_1 x_i) \right\}^2}{2\sigma^2} \right],$$

- The conditional distribution for N y_is given the corresponding x_is can be defined as

$$p(y_1, \ldots, y_N \mid x_1, \ldots, x_N) = \prod_{i=1}^{N} p(y_i \mid x_i).$$

Linear regression as a probability model

- The log-likelihood of our linear model is:

$$L_{y|x} = \log \prod_{i=1}^{N} p(y_i \mid x_i) = \sum_{i=1}^{N} \log p(y_i \mid x_i).$$

$$L_{y|x} = \sum_{i=1}^{N} \log \left\{ \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{\left\{ y_i - (\theta_0 + \theta_1 x_i) \right\}^2}{2\sigma^2} \right] \right\}$$

$$= -N \log\left[\sigma\sqrt{2\pi}\right] - \sum_{i=1}^{N} \frac{\left\{ y_i - (\theta_0 + \theta_1 x_i) \right\}^2}{2\sigma^2}.$$

- To maximize the log-likelihood it suffices to find the parameters that minimize the squared error

$$\arg\max_{\theta_0\ \theta_1}\left(L_{y|x}\right) = \arg\min_{\theta_0\ \theta_1}\left(\sum_{i=1}^{N}\left\{ y_i - (\theta_0 + \theta_1 x_i) \right\}^2\right).$$

i.e. this probability model is simply linear regression

Matrix vector form of regression

- Linear regression can be written in matrix form

$$\sum_{i=1}^{N}\{y_i-(\theta_0+\theta_1 x_i)\}^2=\left(\begin{bmatrix}y_1\\ \vdots\\ y_N\end{bmatrix}-\begin{bmatrix}1 & x_1\\ \vdots & \vdots\\ 1 & x_N\end{bmatrix}\begin{bmatrix}\theta_0\\ \theta_1\end{bmatrix}\right)^T\left(\begin{bmatrix}y_1\\ \vdots\\ y_N\end{bmatrix}-\begin{bmatrix}1 & x_1\\ \vdots & \vdots\\ 1 & x_N\end{bmatrix}\begin{bmatrix}\theta_0\\ \theta_1\end{bmatrix}\right)$$

$$=(\mathbf{y}-\mathbf{A}\mathbf{w})^T(\mathbf{y}-\mathbf{A}\mathbf{w}),$$

- Taking the partial derivative with respect to **w** and setting the result to zero yields a *closed form* expression for the parameters:

$$\frac{\partial}{\partial \mathbf{w}}(\mathbf{y}-\mathbf{A}\mathbf{w})^{\mathrm{T}}(\mathbf{y}-\mathbf{A}\mathbf{w})=0 \quad \Rightarrow \mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{w}=\mathbf{A}^{\mathrm{T}}\mathbf{y}, \quad \mathbf{w}=(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{y}$$

$\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{w}=\mathbf{A}^{\mathrm{T}}\mathbf{y}$ are the famous "normal equations"

$\mathbf{A}^{+}=(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}$ is known as the *pseudoinverse*

Priors on parameters, Gaussians, ridge regression and weight decay

- Placing a zero mean Gaussian prior on the parameters **w** leads to the method of *ridge regression,* also called *weight decay*
- Consider a linear regression that uses a *D* dimensional vector **x** to make predictions
- The regression's bias term can be represented by appending a constant feature = 1 as an additional final dimension of **x** for every example
- The prior is frequently omitted from the bias
- The underlying probability model can be written

$$\prod_{i=1}^{N} p(y_i \mid x_i;\theta)p(\theta;\tau) = \left[\prod_{i=1}^{N} N(y_i;\mathbf{w}^{\mathrm{T}}x_i,\sigma^2)\right]\left[\prod_{d=1}^{D} N(w_d;0,\tau^2)\right]$$

Priors on parameters, L_2 and L_1 regularization

- Maximum a posteriori parameter estimation based on the log conditional likelihood with a zero mean Gaussian prior on weights is equivalent to minimizing a squared error loss function plus an L_2 based regularization term

$$F(\mathbf{w}) = \sum_{i=1}^{N} \left\{ y_i - \mathbf{w}^{\mathrm{T}} x_i \right\}^2 + \lambda R_{L_2}(\mathbf{w}), \quad R_{L_2}(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w} = \left\| \mathbf{w} \right\|_2^2$$

- Using a Laplace prior for the distribution over weights and taking the log of the likelihood function yields an L_1 based regularization term

$$R_{L_1}(\mathbf{w}) = \left\| \mathbf{w} \right\|_1$$

Laplace, L_1 regularization the LASSO and the Elastic Net approach

- The Laplace distribution with params μ and b is

$$P(w;\mu,b) = L(w;\mu,b) = \frac{1}{2b}\exp\left(-\frac{|w-\mu|}{b}\right)$$

- Modeling the log of the prior probability for each weight by a Laplace distribution with $\mu=0$ yields

$$-\log\left[\prod_{d=1}^{D} L(w_d;0,b)\right] = \log(2b) + \frac{1}{b}\sum_{d=1}^{D}|w_d| \propto \|\mathbf{w}\|_1$$

- The use of L_1 regularization is also know as the LASSO, "Least Absolute Shrinkage and Selection Operator".
- An alternative (convex!) approach known as the *elastic net* combines L_1 and L_2 regularization techniques using

$$\lambda_1 R_{L_1}(\boldsymbol{\theta}) + \lambda_2 R_{L_2}(\boldsymbol{\theta}) = \lambda_1 \|\mathbf{w}\|_1 + \lambda_2 \|\mathbf{w}\|_2^2$$

Linear regression with regularization

- For ridge regression our objective function becomes

$$F(\mathbf{w}) = (\mathbf{y} - \mathbf{A}\mathbf{w})^T (\mathbf{y} - \mathbf{A}\mathbf{w}) + \lambda \mathbf{w}^{\mathrm{T}} \mathbf{w}$$

- We get a closed form solution for the result

$$\frac{\partial}{\partial \mathbf{w}} F(\mathbf{w}) = 0$$

$$\Rightarrow \mathbf{A}^{\mathrm{T}} \mathbf{A}\mathbf{w} + \lambda \mathbf{w} = \mathbf{A}^{\mathrm{T}} \mathbf{y}$$

$$\mathbf{w} = (\mathbf{A}^{\mathrm{T}} \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{A}^{\mathrm{T}} \mathbf{y}$$

- This modification to the pseudoinverse equation is very useful, allowing solutions to be found when they would otherwise not exist, often using a very small λ
- The extension to multidimensional inputs **x** can still be formulated using these matrix forms

Polynomial regression and kernels

- We can create a model for non-linear predictions using polynomials of x
- The estimation problem remains linear!

$$\sum_{i=1}^{N}\left\{y_i - (\theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \ldots + \theta_K x_i^K)\right\}^2$$

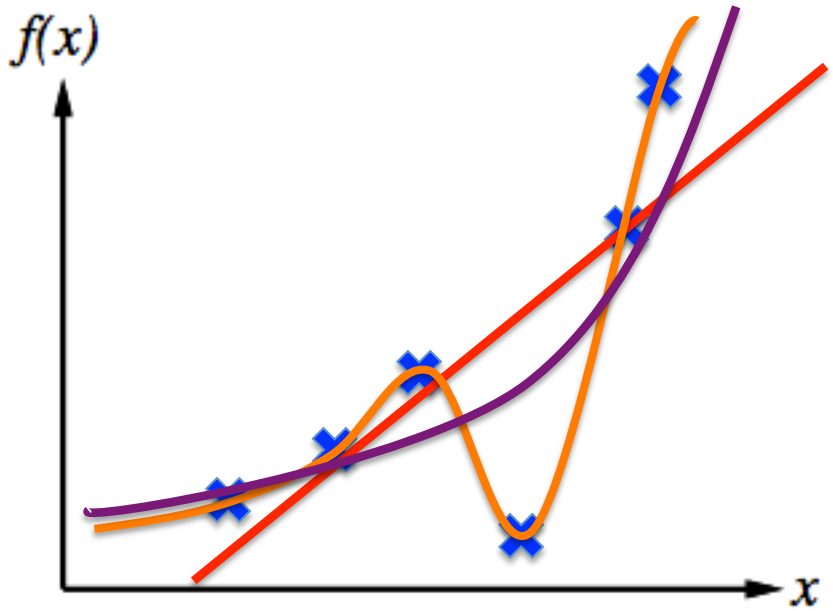
$$= \left(\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^K \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_N & x_N^2 & & x_N^K \end{bmatrix} \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_K \end{bmatrix}\right)^{\mathbf{T}} \left(\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^K \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_N & x_N^2 & & x_N^K \end{bmatrix} \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_K \end{bmatrix}\right)$$

$$= (\mathbf{y} - \mathbf{Ac})^T (\mathbf{y} - \mathbf{Ac}).$$

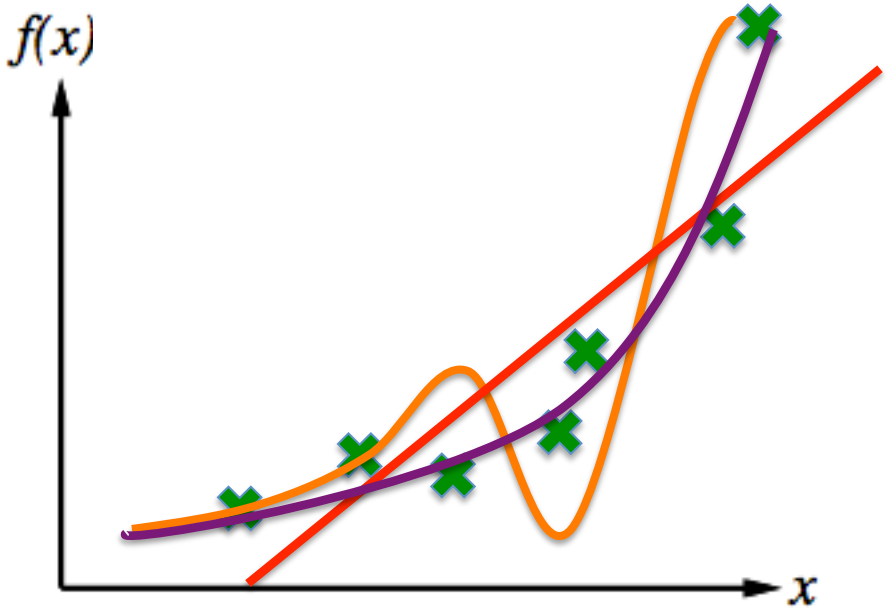
- A similar trick for transforming a linear model into a non-linear model is based on using basis functions $\boldsymbol{\phi}(\mathbf{x})$, where $p(y \mid \mathbf{x}) = N(y; \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}), \sigma^2)$

Question: How do we select the complexity of the model

- How many terms should our polynomial have?
- In general, we use the validation set to make this choice (but there are other approaches).



Training set



Validation set

- Question: could you formulate how to do this as rigorously as possible? (i.e. using math)

Generalized linear models

- Linear regression and logistic regression are special cases of a family of conditional probability models known in statistics as "generalized linear models" (GLMs)
- Idea: unify and generalize linear and logistic regression
- Data can be thought of in terms of response variables y_i and explanatory variables organized as vectors $\mathbf{x}_i$, i=1,…,n.
- Response variables could be expressed in different ways, ranging from binary to categorical or ordinal data
- A model is then defined where μ or the expected value of the distribution used for the response variable consists of an initial linear prediction which is then subjected to a smooth, invertible and potentially non-linear transformation using the *mean function* g^{-1}, i.e.

$$\mu_i = \mathrm{E}[y_i] = g^{-1}(\boldsymbol{\beta}^T \mathbf{x}_i)$$

- The mean function is the inverse of the *link function, g*

GLMs

- In GLMs the entire set of explanatory variables for all the observations is often arranged as a nxp matrix **X**, so that a vector of linear predictions for the entire data set is $\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta}$
- The variance of the underlying distribution can also be modeled; typically as a function of the mean
- Different distributions, link functions and corresponding mean functions give a great deal of flexibility in defining probabilistic models, see examples below

Link Name	**Link Function** $\eta = \beta^T \mathbf{x} = g(\mu)$	**Mean Function** $\mu = g^{-1}(\beta^T \mathbf{x}) = g^{-1}(\eta)$	**Typical Distribution**
Identity	$\eta = \mu$	$\mu = \eta$	Gaussian
Inverse	$\eta = \mu^{-1}$	$\mu = \eta^{-1}$	Exponential
Log	$\eta = \log_e \mu$	$\mu = \exp(\eta)$	Poisson
Log-log	$\eta = -\log(-\log_e \mu)$	$\mu = \exp(-\exp(-\eta))$	Bernoulli
Logit	$\eta = \log_e \frac{\mu}{1-\mu}$	$\mu = \frac{1}{1+\exp(-\eta)}$	Bernoulli
Probit	$\eta = \Phi^{-1}(\mu)$	$\mu = \Phi(\eta)$	Bernoulli

Note: $\Phi(\cdot)$ is the cumulative normal distribution.

Predictions for ordered classes

- To define a model with *M* ordinal categories, *M*–1 *cumulative* probability models of the form $P(Y_i \leq j)$ can be used where the random variable Y_i represents the category of a given instance *i*
- Models for $P(Y_i = j)$ can then be obtained using differences between the cumulative distribution models
- Here we will use complementary cumulative probabilities, known as *survival functions,* of the form $P(Y_i > j) = 1 - P(Y_i \leq j)$
- They sometimes simplify the interpretation of parameters
- The class probabilities can be obtained from:

$$P(Y_i = 1) = 1 - P(Y_i > 1)$$

$$P(Y_i = j) = P(Y_i > j - 1) - P(Y_i > j)$$

$$P(Y_i = M) = P(Y_i > M - 1).$$

- For binary predictions the following model is popular

$$\text{logit}(\gamma_{ij}) = \log \frac{\gamma_{ij}}{1 - \gamma_{ij}} = b_j + \mathbf{w}^T \mathbf{x}_i$$

Conditional probability models based on kernels

- Linear models can be transformed into non-linear ones by applying the "kernel trick"
- Suppose the features **x** are replaced by the vector **k**(**x**) whose elements are determined using a kernel function $k(\mathbf{x},\mathbf{x}_j)$ for every training example:

$$\mathbf{k}(\mathbf{x}) = \begin{bmatrix} k(\mathbf{x},\mathbf{x}_1) \\ \vdots \\ k(\mathbf{x},\mathbf{x}_V) \\ 1 \end{bmatrix}$$

- A "1" has been appended to this vector to implement the bias term in the parameter matrix
- Kernelized regression

$$p(y \mid \mathbf{x}) = N(y; \mathbf{w}^{\mathrm{T}}\mathbf{k}(\mathbf{x}), \sigma^2)$$

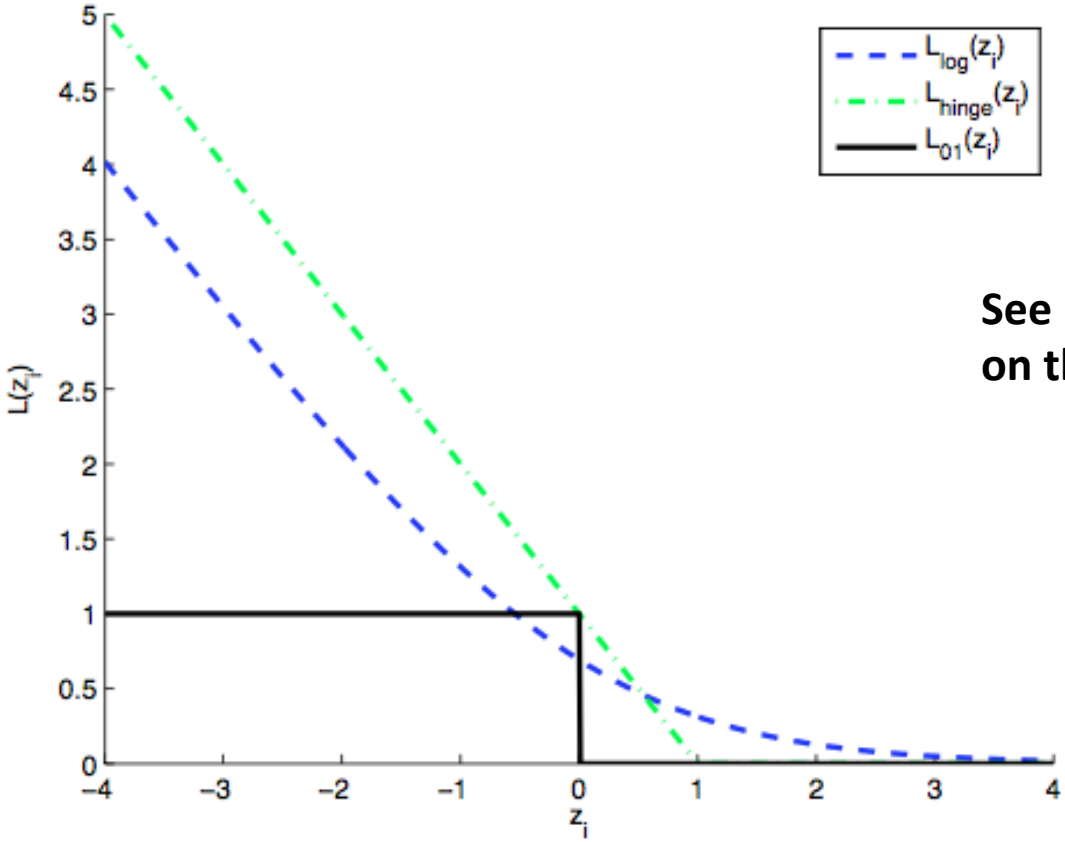
- Kernelized classification

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{\exp\left(\mathbf{y}^T \boldsymbol{\theta}\mathbf{k}(\mathbf{x})\right)}{\sum_{\mathbf{y}} \exp\left(\mathbf{y}^T \boldsymbol{\theta}\mathbf{k}(\mathbf{x})\right)}$$

Losses

See: Md Kamrul Hasan and Christopher Pal. 2019. A New Smooth Approximation to the Zero One Loss with a Probabilistic Interpretation. ACM Trans. Knowl. Discov. Data 14, 1, Article 1 (December 2019), 28 pages. DOI:https://doi.org/10.1145/3365672

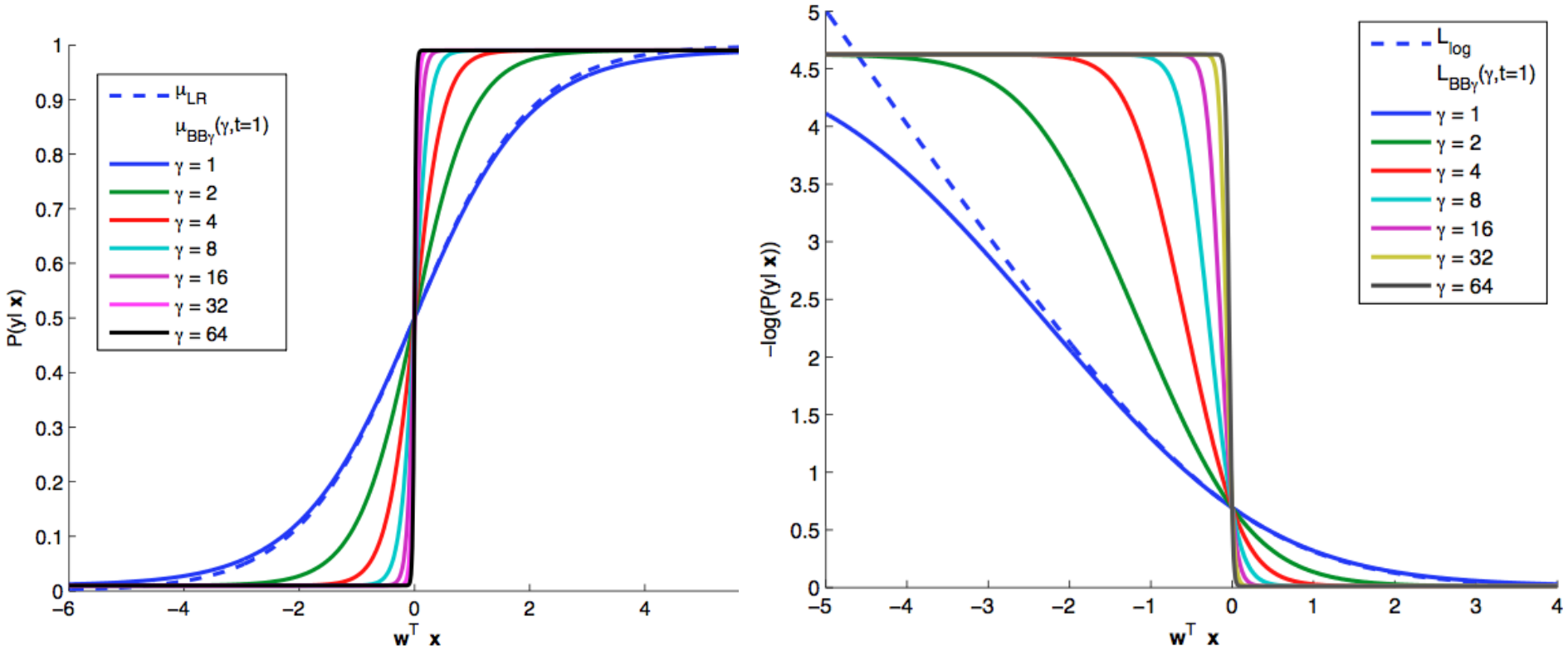
Recent Work
Re-examining the 0-1 Loss



See Hasan & Pal, (2015) on the arXiv

$$L_{log}(z_i) = \log[1 + \exp(-z_i)]$$
$$L_{hinge}(z_i) = \max(0, 1 - z_i)$$
$$L_{01}(z_i) = \mathbb{I}[z_i \leq 0]$$

Can probabilistic methods not 'do this'?



- If our sigmoid for P(y|x) has a *minimal* probability (left), that gives us the smooth approximations to the 01 loss on the right.
- See Hasan and Pal, (2015) *"A New Smooth Approximation to the Zero-One Loss with a Probabilistic Interpretation" on the* arXiv .