Exponential family (cont).
Linear regression

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Exponential family

Exponential family:
• all probability mass / density functions that can be written in the exponential normal form

\[ f(x \mid \eta) = \frac{1}{Z(\eta)} h(x) \exp \left[ \eta^T t(x) \right] \]

• \( \eta \) a vector of natural (or canonical) parameters
• \( t(x) \) a function referred to as a sufficient statistic
• \( h(x) \) a function of \( x \) (it is less important)
• \( Z(\eta) \) a normalization constant (a partition function)

\[ Z(\eta) = \int h(x) \exp \left\{ \eta^T t(x) \right\} dx \]

• Other common form:

\[ f(x \mid \eta) = h(x) \exp \left[ \eta^T t(x) - A(\eta) \right] \quad \log Z(\eta) = A(\eta) \]
### Exponential family: examples

- **Bernoulli distribution**
  \[ p(x \mid \pi) = \pi^x (1 - \pi)^{1-x} \]
  \[ = \exp \left\{ \log \left( \frac{\pi}{1 - \pi} \right) x + \log(1 - \pi) \right\} \]
  \[ = \exp \left\{ \log(1 - \pi) \right\} \exp \left\{ \log \left( \frac{\pi}{1 - \pi} \right) x \right\} \]

- **Exponential family**
  \[ f(x \mid \eta) = \frac{1}{Z(\eta)} h(x) \exp \left[ \eta^T t(x) \right] \]

- **Parameters**
  \[ \eta = ? \quad t(x) = ? \]
  \[ Z(\eta) = ? \quad h(x) = ? \]
Exponential family: examples

- **Univariate Gaussian distribution**
  \[
p(x \mid \mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} \exp\left[-\frac{1}{2\sigma^2} (x - \mu)^2\right]
  = \frac{1}{2\pi} \exp\left(-\frac{\mu}{2\sigma^2} - \log\sigma\right) \exp\left\{\frac{\mu}{\sigma^2} x - \frac{1}{2\sigma^2} x^2\right\}
\]

- **Exponential family**
  \[
f(x \mid \eta) = \frac{1}{Z(\eta)} h(x) \exp[\eta^T t(x)]
\]

- **Parameters**
  \[
  \eta = ? \quad t(x) = ? \\
  Z(\eta) = ? \quad h(x) = ?
\]
Exponential family

- For iid samples, the likelihood of data is

\[ P(D \mid \eta) = \prod_{i=1}^{n} p(x_i \mid \eta) = \prod_{i=1}^{n} h(x_i) \exp \left[ \eta^T t(x_i) - A(\eta) \right] \]

\[ = \left[ \prod_{i=1}^{n} h(x_i) \right] \exp \left[ \sum_{i=1}^{n} \eta^T t(x_i) - A(\eta) \right] \]

\[ = \left[ \prod_{i=1}^{n} h(x_i) \right] \exp \left[ \eta^T \left( \sum_{i=1}^{n} t(x_i) \right) - nA(\eta) \right] \]

- Important:
  - the dimensionality of the sufficient statistic remains the same with the number of samples

---

Exponential family

- The log likelihood of data is

\[ l(D, \eta) = \log \left[ \prod_{i=1}^{n} h(x_i) \right] \exp \left[ \eta^T \left( \sum_{i=1}^{n} t(x_i) \right) - nA(\eta) \right] \]

\[ = \log \left[ \prod_{i=1}^{n} h(x_i) \right] + \left[ \eta^T \left( \sum_{i=1}^{n} t(x_i) \right) - nA(\eta) \right] \]

- Optimizing the loglikelihood

\[ \nabla_\eta l(D, \eta) = \left( \sum_{i=1}^{n} t(x_i) \right) - n \nabla_\eta A(\eta) = 0 \]

- For the ML estimate it must hold

\[ \nabla_\eta A(\eta) = \frac{1}{n} \left( \sum_{i=1}^{n} t(x_i) \right) \]

---

CS 2750 Machine Learning
Exponential family

• Rewriting the gradient:

\[ \nabla_\eta A(\eta) = \nabla_\eta \log Z(\eta) = \nabla_\eta \log \int h(x) \exp \{ \eta^T t(x) \} dx \]

\[ \nabla_\eta A(\eta) = \frac{\int t(x) h(x) \exp \{ \eta^T t(x) \} dx}{\int h(x) \exp \{ \eta^T t(x) \} dx} \]

\[ \nabla_\eta A(\eta) = \int t(x) h(x) \exp \{ \eta^T t(x) - A(\eta) \} dx \]

\[ \nabla_\eta A(\eta) = E(t(x)) \]

• Result:

\[ E(t(x)) = \frac{1}{n} \left( \sum_{i=1}^{n} t(x_i) \right) \]

• For the ML estimate the parameters \( \eta \) should be adjusted such that the expectation of the statistic \( t(x) \) is equal to the observed sample statistics

Moments of the distribution

• For the exponential family

  – The k-th moment of the statistic corresponds to the k-th derivative of \( A(\eta) \)

  – If \( x \) is a component of \( t(x) \) then we get the moments of the distribution by differentiating its corresponding natural parameter

• Example: Bernoulli \( p(x \mid \pi) = \exp \left\{ \log \left( \frac{\pi}{1-\pi} \right) x + \log(1-\pi) \right\} \]

\[ A(\eta) = \log \frac{1}{1-\pi} = \log(1+e^\eta) \]

• Derivatives:

\[ \frac{\partial A(\eta)}{\partial \eta} = \frac{\partial}{\partial \eta} \log(1+e^\eta) = \frac{e^\eta}{1+e^\eta} = \frac{1}{(1+e^{-\eta})} = \pi \]

\[ \frac{\partial A(\eta)}{\partial \eta^2} = \frac{\partial}{\partial \eta} \frac{1}{1+e^{-\eta}} = \pi(1-\pi) \]
Outline

Linear Regression
- Linear model
- Error function based on the least squares fit
- Parameter estimation.
- Gradient methods.
- On-line regression techniques.
- Linear additive models
- Statistical model of linear regression

Supervised learning

Data: \( D = \{D_1, D_2, \ldots, D_n\} \) a set of \( n \) examples

\[ D_i = \langle x_i, y_i \rangle \]

\( x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d}) \) is an input vector of size \( d \)

\( y_i \) is the desired output (given by a teacher)

Objective: learn the mapping \( f: X \rightarrow Y \)

\( s.t. \quad y_i \approx f(x_i) \quad \text{for all} \quad i = 1, \ldots, n \)

- Regression: \( Y \) is continuous
  Example: earnings, product orders \( \rightarrow \) company stock price
- Classification: \( Y \) is discrete
  Example: handwritten digit in binary form \( \rightarrow \) digit label
Linear regression

- **Function** $f : X \rightarrow Y$ is a linear combination of input components

  \[
f(x) = w_0 + w_1x_1 + w_2x_2 + \ldots + w_dx_d = w_0 + \sum_{j=1}^{d} w_jx_j
  \]

  $w_0, w_1, \ldots, w_k$ - parameters (weights)

- **Shorter (vector) definition of the model**
  
  - Include bias constant in the input vector
  
  $\mathbf{x} = (1, x_1, x_2, \ldots, x_d)$

  \[
f(x) = w_0x_0 + w_1x_1 + w_2x_2 + \ldots + w_dx_d = \mathbf{w}^T \mathbf{x}
  \]

  $w_0, w_1, \ldots, w_k$ - parameters (weights)
Linear regression. Error.

- **Data:** $D_i = \langle x_i, y_i \rangle$
- **Function:** $x_i \rightarrow f(x_i)$
- We would like to have $y_i \approx f(x_i)$ for all $i = 1, ..., n$

- **Error function**
  - measures how much our predictions deviate from the desired answers
  
  $J_n = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$

- **Learning:**
  We want to find the weights minimizing the error!

Linear regression. Example

- 1 dimensional input $x = (x_1)$

![Graph of linear regression example]
Linear regression. Example.

- 2 dimensional input \( \mathbf{x} = (x_1, x_2) \)

![Plot](image)

Linear regression. Optimization.

- We want the **weights minimizing the error**
  
  \[
  J_n = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2
  \]

- For the optimal set of parameters, derivatives of the error with respect to each parameter must be 0

\[
\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,j} = 0
\]

- **Vector of derivatives:**

\[
\nabla \mathbf{w} (J_n(\mathbf{w})) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i = \mathbf{0}
\]
Linear regression. Optimization.

- grad \( w \) \((J_n(w)) = 0\) defines a set of equations in \( w \)

\[
\frac{\partial}{\partial w_0} J_n(w) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) = 0
\]

\[
\frac{\partial}{\partial w_1} J_n(w) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,1} = 0
\]

\[
\vdots
\]

\[
\frac{\partial}{\partial w_j} J_n(w) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,j} = 0
\]

\[
\vdots
\]

\[
\frac{\partial}{\partial w_d} J_n(w) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,d} = 0
\]

---

Solving linear regression

\[
\frac{\partial}{\partial w_j} J_n(w) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,j} = 0
\]

By rearranging the terms we get a system of linear equations with \( d+1 \) unknowns

\[
Aw = b
\]

\[
w_0 \sum_{i=1}^{n} x_{i,0} + w_1 \sum_{i=1}^{n} x_{i,1} + \ldots + w_j \sum_{i=1}^{n} x_{i,j} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} = \sum_{i=1}^{n} y_i
\]

\[
w_0 \sum_{i=1}^{n} x_{i,0} x_{i,1} + w_1 \sum_{i=1}^{n} x_{i,1} x_{i,1} + \ldots + w_j \sum_{i=1}^{n} x_{i,j} x_{i,j} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} x_{i,d} = \sum_{i=1}^{n} y_i x_{i,1}
\]

\[
\vdots
\]

\[
w_0 \sum_{i=1}^{n} x_{i,0} x_{i,j} + w_1 \sum_{i=1}^{n} x_{i,1} x_{i,j} + \ldots + w_j \sum_{i=1}^{n} x_{i,j} x_{i,j} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} x_{i,j} = \sum_{i=1}^{n} y_i x_{i,j}
\]

\[
\vdots
\]

---

CS 2750 Machine Learning
Solving linear regression

- The optimal set of weights satisfies:
  \[ \nabla_w (J_n (w)) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w^T x_i) x_i = 0 \]

  Leads to a **system of linear equations (SLE)** with \( d+1 \) unknowns of the form

  \[ A w = b \]

  \[ w_0 \sum_{i=1}^{n} x_{i,0} x_{i,j} + w_1 \sum_{i=1}^{n} x_{i,1} x_{i,j} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} x_{i,j} = \sum_{i=1}^{n} y_i x_{i,j} \]

  **Solution to SLE:** ?

---

Solving linear regression

- The optimal set of weights satisfies:
  \[ \nabla_w (J_n (w)) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w^T x_i) x_i = 0 \]

  Leads to a **system of linear equations (SLE)** with \( d+1 \) unknowns of the form

  \[ A w = b \]

  \[ w_0 \sum_{i=1}^{n} x_{i,0} x_{i,j} + w_1 \sum_{i=1}^{n} x_{i,1} x_{i,j} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} x_{i,j} = \sum_{i=1}^{n} y_i x_{i,j} \]

  **Solution to SLE:** \[ w = A^{-1} b \]

  - matrix inversion
Gradient descent solution

**Goal:** the weight optimization in the linear regression model

\[ J_n = \text{Error}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i, \mathbf{w}))^2 \]

An alternative to SLE solution:

- **Gradient descent**
  
  **Idea:**
  
  - Adjust weights in the direction that improves the Error
  - The gradient tells us what is the right direction

  \[ \mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \text{Error}_i(\mathbf{w}) \]

  \( \alpha > 0 \) - a **learning rate** (scales the gradient changes)

Gradient descent method

- Descend using the gradient information

\[ \text{Error}(\mathbf{w}) \]

\[ \nabla_{\mathbf{w}} \text{Error}(\mathbf{w}) \big|_{\mathbf{w}^*} \]

- Change the value of \( \mathbf{w} \) according to the gradient

\[ \mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \text{Error}_i(\mathbf{w}) \]
Gradient descent method

- New value of the parameter

\[ w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} \text{Error} (w) \bigg|_{w^*} \quad \text{For all } j \]

\[ \alpha > 0 \] - a learning rate (scales the gradient changes)

Gradient descent method

- Iteratively approaches the optimum of the Error function
Online gradient algorithm

- The error function is defined for the whole dataset $D$
  \[ J_n = Error(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i, w))^2 \]
- error for a sample $D_i = <x_i, y_i>$
  \[ J_{\text{online}} = Error_i(w) = \frac{1}{2} (y_i - f(x_i, w))^2 \]
- Online gradient method: changes weights after every sample
  \[ w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} Error_i(w) \]
- vector form:
  \[ w \leftarrow w - \alpha \nabla_w Error_i(w) \]
  \[ \alpha > 0 \quad \text{- Learning rate that depends on the number of updates} \]

Online gradient method

Linear model
\[ f(x) = w^T x \]
On-line error
\[ J_{\text{online}} = Error_i(w) = \frac{1}{2} (y_i - f(x_i, w))^2 \]
On-line algorithm: generates a sequence of online updates

(i)-th update step with: $D_i = <x_i, y_i>$

j-th weight:
\[ w_j^{(i)} \leftarrow w_j^{(i-1)} - \alpha(i) \frac{\partial Error_i(w)}{\partial w_j} \bigg|_{w^{(i-1)}} \]
\[ w_j^{(i)} \leftarrow w_j^{(i-1)} + \alpha(i)(y_i - f(x_i, w^{(i-1)}))x_{i,j} \]

Fixed learning rate: $\alpha(i) = C$  
Annealed learning rate: $\alpha(i) \approx \frac{1}{i}$
- Use a small constant  
- Gradually rescales changes
Online regression algorithm

**Online-linear-regression** \((D, \text{number of iterations})\)

Initialize weights \(w = (w_0, w_1, w_2 \ldots w_d)\)

for \(i = 1:1: \text{number of iterations} \)

- select a data point \(D_i = (x_i, y_i)\) from \(D\)
- set learning rate \(\alpha(i)\)
- update weight vector
  \[ w \leftarrow w + \alpha(i)(y_i - f(x_i, w))x_i \]

end for

return weights \(w\)

- **Advantages:** very easy to implement, continuous data streams
Practical concerns: Input normalization

• Input normalization
  – makes the data vary roughly on the same scale.
  – Can make a huge difference in on-line learning

Assume on-line update (delta) rule for two weights $j,k,$:

\[
\begin{align*}
  w_j &\leftarrow w_j + \alpha(i)(y_i - f(x_i))x_{i,j} \\
  w_k &\leftarrow w_k + \alpha(i)(y_i - f(x_i))x_{i,k}
\end{align*}
\]

Change depends on the magnitude of the input.

For inputs with a large magnitude the change in the weight is huge: changes to the inputs with high magnitude disproportional as if the input was more important.

Input normalization

• Input normalization:
  – Solution to the problem of different scales
  – Makes all inputs vary in the same range around 0

\[
\begin{align*}
  \bar{x}_j &= \frac{1}{n} \sum_{i=1}^{n} x_{i,j} \\
  \sigma_j^2 &= \frac{1}{n-1} \sum_{i=1}^{n} (x_{i,j} - \bar{x}_j)^2
\end{align*}
\]

New input: \( \tilde{x}_{i,j} = \frac{(x_{i,j} - \bar{x}_j)}{\sigma_j} \)

More complex normalization approach can be applied when we want to process data with correlations

Similarly we can renormalize outputs $y$
Extensions of simple linear model

Replace inputs to linear units with feature (basis) functions to model **nonlinearities**

\[ f(x) = w_0 + \sum_{j=1}^{m} w_j \phi_j(x) \]

\( \phi_j(x) \) - an arbitrary function of \( x \)

The same techniques as before to learn the weights