Logistic regression

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

Announcements

Homework 10:  
• due on Thursday, December 2, 2004

Final exam:  December 14, 2004 at 1:00pm-3:00pm
• Location: Sennott Square 5129
• Closed book
• Cumulative
Supervised learning

Data: \( D = \{ D_1, D_2, ..., D_n \} \) a set of \( n \) examples
\[ D_i = \langle x_i, y_i \rangle \]
\( x_i = (x_{i,1}, x_{i,2}, ..., x_{i,d}) \) is an input vector of size \( d \)
\( y_i \) is the desired output (given by a teacher)

Objective: learn the mapping \( f : X \rightarrow Y \)
\[ s.t. \quad y_i \approx f(x_i) \quad \text{for all } i = 1, ..., n \]

- **Regression**: \( Y \) is **continuous**
  Example: earnings, product orders \( \rightarrow \) company stock price
- **Classification**: \( Y \) is **discrete**
  Example: handwritten digit \( \rightarrow \) digit label

---

Linear regression

- **Function** \( f : X \rightarrow Y \) is a linear combination of input components
\[ f(x) = w_0 + w_1 x_1 + w_2 x_2 + ... + w_d x_d = w_0 + \sum_{j=1}^{d} w_j x_j \]

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

• **Shorter (vector) definition of the model**
  – Include bias constant in the input vector
    \[ x = (1, x_1, x_2, \ldots, x_d) \]

    \[ f(x) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d = \mathbf{w}^T \mathbf{x} \]

    \[ w_0, w_1, \ldots, w_k \text{ - parameters (weights)} \]

\[ \sum \]

\[ f(\mathbf{x}, \mathbf{w}) \]

Input vector \[ \mathbf{x} \]

\[
\begin{align*}
1 & \quad w_0 \\
 x_1 & \quad w_1 \\
 x_2 & \quad w_2 \\
 \vdots & \\
 x_d & \quad w_d
\end{align*}
\]

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 \( i = 1, \ldots, n \)

• **Error function**
  – measures how much our predictions deviate from the desired answers

  \[
  \text{Mean-squared error} \quad J_n = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2
  \]

• **Learning**:
  We want to find the weights minimizing the error!
Linear regression. Example

- 1 dimensional input \( x = (x_1) \)

- 2 dimensional input \( x = (x_1, x_2) \)
Linear regression. Optimization.

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

- For the optimal set of parameters, derivatives of the error with respect to each parameter must be 0
  \[ \frac{\partial}{\partial w_j} J_n(w) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,j} = 0 \]

- Vector of derivatives:
  \[ \text{grad}_w (J_n(w)) = \nabla_w (J_n(w)) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w^T x_i) x_i = \mathbf{0} \]

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

\[
\begin{align*}
Aw &= b \\
\end{align*}
\]

\[
\begin{align*}
w_0 \sum_{i=1}^{n} x_{i,0} + w_1 \sum_{i=1}^{n} x_{i,1} + \ldots + w_j \sum_{i=1}^{n} x_{i,j} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} &= \sum_{i=1}^{n} y_i \\
w_0 \sum_{i=1}^{n} x_{i,0} x_{i,1} + w_1 \sum_{i=1}^{n} x_{i,1} x_{i,1} + \ldots + w_j \sum_{i=1}^{n} x_{i,j} x_{i,j} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} x_{i,j} &= \sum_{i=1}^{n} y_i x_{i,j} \\
\cdots \\
w_0 \sum_{i=1}^{n} x_{i,0} x_{i,j} + w_1 \sum_{i=1}^{n} x_{i,1} x_{i,j} + \ldots + w_j \sum_{i=1}^{n} x_{i,j} x_{i,j} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} x_{i,j} &= \sum_{i=1}^{n} y_i x_{i,j} \\
\cdots 
\end{align*}
\]
**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
  \[ Aw = b \]

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

- matrix inversion

**Gradient descent solution**

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

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

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

  \[ w \leftarrow w - \alpha \nabla_w \text{Error}_n(w) \]

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

- Descend using the gradient information

\[ \nabla_w \text{Error} \left( w \right) \]

\[ w^* \]

Direction of the descent

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

\[ w \leftarrow w - \alpha \nabla_w \text{Error} \left( w \right) \]

New value of the parameter

\[ w_j \leftarrow w_j^* + \alpha \frac{\partial}{\partial w_j} \text{Error} \left( w \right) \]

For all \( j \)

\( \alpha > 0 \) - a learning rate (scales the gradient changes)
Gradient descent method

- Iteratively converge to the optimum of the Error function

\[ Error(w) \]

\[ w^{(0)} \rightarrow w^{(1)} \rightarrow w^{(2)} \rightarrow w^{(3)} \]

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 \( D_i \)
  \[ D_i = \langle x_i, y_i \rangle \]
  \[ J_{\text{online}} = Error_i(w) = \frac{1}{2} (y_i - f(x_i, w))^2 \]

- Online gradient method: changes weights after every sample
  \[ \begin{align*}
  w_j &\leftarrow w_j - \alpha \frac{\partial}{\partial w_j} Error_i(w) \\
  w &\leftarrow w - \alpha \nabla_w Error_i(w)
  \end{align*} \]

\[ \alpha > 0 \quad - \text{Learning rate that depends on the number of updates} \]
Online gradient method

Linear model \( f(x) = w^T x \)

On-line error \( J_{\text{online}} = \text{Error}_i(w) = \frac{1}{2}(y_i - f(x_i, w))^2 \)

On-line algorithm: generates a sequence of online updates

(i)-th update step with: \( D_i = \langle x_i, y_i \rangle \)

j-th weight:

\[
 w_j^{(i)} \leftarrow w_j^{(i-1)} - \alpha(i) \frac{\partial \text{Error}_i(w)}{\partial w_j}\bigg|_{w^{(i-1)}}
\]

\[
 w_j^{(i)} \leftarrow w_j^{(i-1)} + \alpha(i)(y_i - f(x_i, w^{(i-1)}))x_{i,j}
\]

Annealed learning rate: \( \alpha(i) \approx \frac{1}{i} \)
- Gradually rescales changes in weights

Online regression algorithm

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

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

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

\( \text{do} \)

\( \text{select} \) a data point \( D_i = (x_i, y_i) \) from \( D \)

\( \text{set} \) \( \alpha = \frac{1}{i} \)

\( \text{update} \) weight vector

\( w \leftarrow w + \alpha(y_i - f(x_i, w))x_i \)

end for

return weights \( w \)

• Advantages: very easy to implement, continuous data streams, adapts to changes in the model over time
On-line learning. Example

Logistic regression
Binary classification

- Two classes \( Y = \{0, 1\} \)
- Our goal is to learn how to correctly classify two types of examples
  - Class 0 – labeled as 0,
  - Class 1 – labeled as 1

- We would like to learn \( f : X \rightarrow \{0, 1\} \)

- First step: we need to devise a model of the function \( f \)

- Inspiration: neuron (nerve cells)
Neuron-based binary classification model

Neuron-based binary classification

• Function we want to learn $f : X \rightarrow \{0, 1\}$
Logistic regression model

- A function model with smooth switching:
  \[ f(x) = g \left( w_0 + w_1x^{(1)} + ... + w_kx^{(k)} \right) \]
  where \( w \) are parameters of the models
  and \( g(z) \) is a **logistic function** \( g(z) = \frac{1}{1 + e^{-z}} \)

Bias term \[ k \]
Input vector \[ x \]
Logistic function
\[ f(x) \in [0,1] \]

Logistic function

- also referred to as **sigmoid function**
- **replaces threshold function** with smooth switching
- takes a real number and outputs the number in the interval \([0,1]\)
- Output of the logistic regression has a probabilistic interpretation
  \[ f(x) = p(y = 1 | x) \quad - \text{Probability of class 1} \]
Logistic regression. Decision boundary

Classification with the logistic regression model:

If \( f(x) \geq 1/2 \) then choose class 1
Else choose class 0

Logistic regression model defines a linear decision boundary

Example:

Optimization of weights

- **Two classes:** \( Y = \{0, 1\} \)
- **Data:** \( D = \{d_1, d_2, \ldots, d_n\} \)
  \( d_i = \langle x_i, y_i \rangle \)
- We want to find the set of weight \( w \) that explain the data the best
- **Zero-one error function**
  \[
  Error(x_i, y_i) = \begin{cases} 
  1 & f(x_i, w) \neq y_i \\
  0 & f(x_i, w) = y_i
  \end{cases}
  \]
- Error we would like to minimize: \( E_{(x,y)}(Error(x,y)) \)
- The error is minimized if we choose:
  \[
  y = 1 \text{ if } p(y = 1 \mid x, w) > p(y = 0 \mid x, w) \\
  y = 0 \text{ otherwise}
  \]
Logistic regression. Parameter optimization.

- We construct a probabilistic version of the **error function** based on the **likelihood of the data**
  \[ L(D, w) = P(D \mid w) = \prod_{i=1}^{n} P(y_i = y_i \mid x_i, w) \]
  
  Independent samples \((x_i, y_i)\)

- likelihood measures the goodness of fit (opposite of the error)

- Log-likelihood trick.
  \[ l(D, w) = \log L(D \mid w) \]

- Error is the opposite of the Log likelihood
  \[ Error(D, w) = -l(D, w) \]

---

Logistic regression: parameter learning

- **Error function decomposes to online error components**
  \[ Error(D, w) = \sum_{i=1}^{n} Error_i(D_i, w) = -\sum_{i=1}^{n} l_i(D_i, w) \]

- Derivatives of the online error component for the LR model (in terms of weights)
  \[ \frac{\partial}{\partial w_0} Error_i(D_i, w) = -(y_i - f(x_i, w)) \]
  \[ \ddots \]
  \[ \frac{\partial}{\partial w_j} Error_i(D_i, w) = -(y_i - f(x_i, w))x_{i,j} \]
Logistic regression: parameter learning.

- **Assume** \( D_i = \langle x_i, y_i \rangle \)
- **Let**
  \[ \mu_i = p(y_i = 1 \mid x_i, w) = g(z_i) = g(w^T x) \]
- **Then**
  \[ L(D, w) = \prod_{i=1}^{n} P(y = y_i \mid x_i, w) = \prod_{i=1}^{n} \mu_i^{y_i}(1 - \mu_i)^{1-y_i} \]
- **Find weights** \( w \) **that maximize the likelihood of outputs**
  - The optimal weights are the same for both the likelihood and the log-likelihood
  \[ l(D, w) = \log \prod_{i=1}^{n} \mu_i^{y_i}(1 - \mu_i)^{1-y_i} = \sum_{i=1}^{n} \log \mu_i^{y_i}(1 - \mu_i)^{1-y_i} = \]
  \[ = \sum_{i=1}^{n} y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) = \sum_{i=1}^{n} -J_{\text{online}}(D_i, w) \]

Logistic regression: parameter learning

- **Log likelihood**
  \[ l(D, w) = \sum_{i=1}^{n} y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \]
- **Derivatives of the loglikelihood**
  \[ -\frac{\partial}{\partial w_j} l(D, w) = \sum_{i=1}^{n} -x_{i,j}(y_i - g(z_i)) \text{ Nonlinear in weights !!} \]
  \[ \nabla_w -l(D, w) = \sum_{i=1}^{n} -x_i(y_i - g(w^T x_i)) = \sum_{i=1}^{n} -x_i(y_i - f(w, x_i)) \]
- **Gradient descent:**
  \[ w^{(k)} \leftarrow w^{(k-1)} - \alpha(k) \nabla_w [\sum_{i=1}^{n} -l(D, w)] |_{w^{(k-1)}} \]
  \[ w^{(k)} \leftarrow w^{(k-1)} + \alpha(k) \sum_{i=1}^{n} [y_i - f(w^{(k-1)}, x_i)] x_i \]
Derivation of the gradient

- **Log likelihood** \( l(D, w) = \sum_{i=1}^{n} y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \)

- **Derivatives of the loglikelihood**
  \[
  \frac{\partial}{\partial w_j} l(D, w) = \sum_{i=1}^{n} \frac{\partial}{\partial z_i} \left[ y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \right] \frac{\partial z_i}{\partial w_j}
  \]
  
  Derivative of a logistic function
  
  \[
  \frac{\partial g(z_i)}{\partial z_i} = g(z_i)(1 - g(z_i))
  \]

  \[
  \frac{\partial}{\partial z_i} [y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)] = y_i - \frac{1}{g(z_i)} \frac{\partial g(z_i)}{\partial z_i} + (1 - y_i) \frac{-1}{1 - g(z_i)} \frac{\partial g(z_i)}{\partial z_i}
  \]

  
  \[
  = y_i(1 - g(z_i)) + (1 - y_i)(-g(z_i)) = y_i - g(z_i)
  \]

  \[
  \nabla_w l(D, w) = \sum_{i=1}^{n} -x_i (y_i - g(w^T x_i)) = \sum_{i=1}^{n} -x_i (y_i - f(w, x_i))
  \]

Logistic regression. Online gradient.

- We want to optimize the online Error

- **On-line gradient update for the jth weight and ith step**
  \[
  w_j^{(i)} \leftarrow w_j^{(i-1)} - \alpha \frac{\partial}{\partial w_j} [\text{Error}_i(D_i, w) |_{w^{(i-1)}}]
  \]

- **(i)th update for the logistic regression** \( D_i = \langle x_i, y_i \rangle \)

  J-th weight

  \[
  w_j^{(i)} \leftarrow w_j^{(i-1)} + \alpha (i) \left(y_i - f(x_i, w_j^{(i-1)})\right) x_{i,j}
  \]

  \( \alpha \) - annealed learning rate (depends on the number of updates)

  **The same update rule as used in the linear regression !!!**
Online logistic regression algorithm

**Online-logistic-regression** (*D, number of iterations*)

- **initialize** weights $w_0, w_1, w_2 \ldots w_k$
- **for** $i = 1:1: number of iterations$
  - **do**
    - select a data point $d = <x, y>$ from $D$
    - set $\alpha = 1/i$
    - **update** weights (in parallel)
      
      $w_0 = w_0 + \alpha[y - f(x, w)]$
      
      $\vdots$
      
      $w_j = w_j + \alpha[y - f(x, w)]x_j$
  - **end for**
- **return** weights

Online algorithm. Example.
Online algorithm. Example.