Logistic regression

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. \( y_i \approx f(x_i) \) for all \( i = 1, ..., n \)

- **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: review

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

\[
 f(x) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d = w_0 + \sum_{j=1}^{d} w_j x_j
\]

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

**Bias term** \( \rightarrow 1 \)

**Input vector** \( \{x_1, x_2, \ldots, x_d\} \)

\( f(x, w) \)

\[ f(x) = \sum_{j=1}^{d} w_j x_j \]

Linear regression: review

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

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

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

- **Learning:**
  We want to find the weights minimizing the error!
Solving linear regression: review

• 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

\[ \mathbf{A} \mathbf{w} = \mathbf{b} \]

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

**Solutions to SLE:**
• e.g. matrix inversion (if the matrix is singular)

\[ \mathbf{w} = \mathbf{A}^{-1} \mathbf{b} \]

---

Linear regression. Example

• 1 dimensional input \( \mathbf{x} = (x_1) \)
Linear regression. Example.

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

![Graph showing linear regression example](image)

**Gradient descent solution**

- There are other ways to solve the weight optimization problem in the linear regression model

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

- A simple technique:
  - **Gradient descent**

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

\[
w \leftarrow w - \alpha \nabla_w Error_i(w)
\]

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

- Descend using the gradient information

$$\nabla_w \text{Error}(w) \big|_{w^*}$$

Direction of the descent

- Change the value of $w$ according to the gradient

$$w \leftarrow w - \alpha \nabla_w \text{Error}_f(w)$$

New value of the parameter

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

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

- Iteratively converge to the optimum of the Error function

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

\[
\begin{align*}
\mathbf{w}^{(0)} & \to \mathbf{w}^{(1)} & \to \mathbf{w}^{(2)} & \to \mathbf{w}^{(3)} \\
\end{align*}
\]

Online regression algorithm

- **Batch learning**: error function defined for the whole dataset \( D \)
  \[
  J_n = \text{Error}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i, \mathbf{w}))^2
  \]

- **On-line learning**: Error function for individual datapoints.
  - Useful when we learn on datastreams
  \[
  D_i = <x_i, y_i>
  \]
  \[
  J_{\text{online}} = \text{Error}_i(\mathbf{w}) = \frac{1}{2} (y_i - f(x_i, \mathbf{w}))^2
  \]

- Change regression weights after every example according to the gradient of the online error:
  \[
  \mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \text{Error}_i(\mathbf{w})
  \]
Gradient for on-line learning

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:** sequence of online updates

(i)-th update for the linear model: \( D_i = \langle x_i, y_i \rangle \)

Vector form:
\[
w^{(i)} \leftarrow w^{(i-1)} - \alpha(i) \nabla_w \text{Error}_i(w) \big|_{w^{(i-1)}} = w^{(i-1)} + \alpha(i) (y_i - f(x_i, w^{(i-1)}))x_i
\]

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

**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** \( i=1:1 \): **number of iterations**

**do**

- **select** a data point \( D_i = (x_i, y_i) \) from \( D \)
- **set** \( \alpha = 1/i \)

**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
On-line learning. Example

Supervised learning

Data: \( D = \{ d_1, d_2, \ldots, d_n \} \)  a set of \( n \) examples

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

\( x_i \) is input vector, and \( y \) is desired output (given by a teacher)

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

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

Two types of problems:

- **Regression**: \( Y \) is continuous
  - Example: earnings, product orders \( \rightarrow \) company stock price
- **Classification**: \( Y \) is discrete
  - Example: temperature, heart rate \( \rightarrow \) disease

Today: **binary classification problems**
Binary classification

- **Two classes** \( Y = \{0,1\} \)
- Our goal is to learn to classify correctly two types of examples
  - Class 0 – labeled as 0,
  - Class 1 – labeled as 1
- We would like to learn \( f : X \rightarrow \{0,1\} \)
- **Zero-one error (loss) function**
  \[
  Error_i(x_i, y_i) = \begin{cases} 
  1 & f(x_i, w) \neq y_i \\
  0 & f(x_i, w) = y_i
  \end{cases}
  \]
- Error we would like to minimize: \( E_{(x,y)}(Error_i(x,y)) \)
- **First step:** we need to devise a model of the function

---

Discriminant functions

- One convenient way to represent classifiers is through
  - **Discriminant functions**
- Works for binary and multi-way classification

- **Idea:**
  - For every class \( i = 0,1, \ldots, k \) define a function \( g_i(x) \)
    - mapping \( X \rightarrow \mathbb{R} \)
  - When the decision on input \( x \) should be made choose the
    class with the highest value of \( g_i(x) \)

- So what happens with the input space? Assume a binary case.
Discriminant functions

- Define decision boundary.

\[ g_1(x) \geq g_0(x) \]

\[ g_1(x) = g_0(x) \]

\[ g_1(x) \leq g_0(x) \]
Logistic regression model

- Defines a linear decision boundary
- Discriminant functions:
  \[ g_1(x) = g(w^T x) \quad g_0(x) = 1 - g(w^T x) \]
- where \( g(z) = \frac{1}{1 + e^{-z}} \) - is a logistic function

\[ f(x, w) = g_1(w^T x) = g(w^T x) \]

\[ z = \sum \begin{cases} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{cases} w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d \]

Logistic function

function

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

- also referred to as a sigmoid function
- Replaces the threshold function with smooth switching
- takes a real number and outputs the number in the interval [0,1]
Logistic regression model

- **Discriminant functions:**
  \[
g_1(x) = g(w^T x) \quad g_0(x) = 1 - g(w^T x)
\]
- **Where** \( g(z) = 1/(1 + e^{-z}) \) - is a logistic function
- **Values of discriminant functions vary in [0,1]**
  - **Probabilistic interpretation**
    \[
f(x, w) = p(y = 1 \mid w, x) = g_1(x) = g(w^T x)
\]

![Diagram of Logistic Regression](image)

Logistic regression

- Instead of learning the mapping to discrete values 0,1
  \[
f : X \to \{0,1\}
\]
- we learn a **probabilistic function**
  \[
f : X \to [0,1]
\]
  - where \( f \) describes the probability of class 1 given \( x \)
    \[
f(x, w) = p(y = 1 \mid x, w)
\]
  **Note that:** \( p(y = 0 \mid x, w) = 1 - p(y = 1 \mid x, w) \)
- **Transformation to discrete class values:**
  
  If \( p(y = 1 \mid x) \geq 1/2 \) then choose 1
  Else choose 0
Linear decision boundary

- Logistic regression model defines a linear decision boundary
- Why?
- **Answer:** Compare two discriminant functions.
- Decision boundary: \( g_1(x) = g_0(x) \)
- For the boundary it must hold:
  \[
  \log \frac{g_0(x)}{g_1(x)} = \log \frac{1 - g(w^T x)}{g(w^T x)} = 0
  \]
  \[
  \log \frac{g_0(x)}{g_1(x)} = \log \frac{\exp(-w^T x)}{1 + \exp(-w^T x)} = \log \exp(-w^T x) = w^T x = 0
  \]

---

Logistic regression model. Decision boundary

- **LR defines a linear decision boundary**
  **Example:** 2 classes (blue and red points)
Logistic regression: parameter learning.

Likelihood of outputs
• Let
  \[ D_i = \langle x_i, y_i \rangle \quad \mu_i = p(y_i = 1 \mid x_i, w) = g(z_i) = g(w^T x) \]
• 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
  – Apply the log-likelihood trick The optimal weights are the same for both the likelihood and the log-likelihood
  \[ l(D, w) = \log \prod_{i=1}^{n} \mu_i^{y_i} (1 - \mu_i)^{1-y_i} = \sum_{i=1}^{n} \log \mu_i^{y_i} (1 - \mu_i)^{1-y_i} = \]
  \[ = \sum_{i=1}^{n} y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \]

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
  \[ -\frac{\partial}{\partial w_j} l(D, w) = \sum_{i=1}^{n} -x_{i,j} (y_i - g(z_i)) \]
  \[ \nabla_w -l(D, w) = \sum_{i=1}^{n} -x_i (y_i - g(w^T x_i)) = \sum_{i=1}^{n} -x_i (y_i - f(w, x_i)) \]
• Gradient descent:
  \[ w^{(k)} \leftarrow w^{(k-1)} - \alpha(k) \nabla_w [-l(D, w)] \bigg|_{w^{(k-1)}} \]
  \[ = w^{(k-1)} + \alpha(k) \sum_{i=1}^{n} [y_i - f(w^{(k-1)}, x_i)] x_i \]
Logistic regression. Online gradient descent

- **On-line component of the loglikelihood**
  
  \(-J_{\text{online}}(D_i, \mathbf{w}) = y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)\)

- **On-line learning update for weight \(w\)**
  
  \[w^{(k)} \leftarrow w^{(k-1)} - \alpha(k) \nabla_w [J_{\text{online}}(D_k, \mathbf{w})] |_{w^{(k-1)}}\]

- **ith update for the logistic regression** and \(D_k = \langle x_k, y_k \rangle\)
  
  \[w^{(i)} \leftarrow w^{(k-1)} + \alpha(k) [y_i - f(w^{(k-1)}, x_k)]x_k\]

---

Online logistic regression algorithm

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

initialize weights \(\mathbf{w} = (w_0, w_1, w_2 \ldots w_d)\)

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

  do select a data point \(D_i = \langle x_i, y_i \rangle\) from \(D\)

  set \(\alpha = 1/i\)

  update weights (in parallel)

  \[\mathbf{w} \leftarrow \mathbf{w} + \alpha(i) [y_i - f(\mathbf{w}, x_i)]x_i\]

end for

return weights \(\mathbf{w}\)
Online algorithm. Example.
Online algorithm. Example.

Derivation of the gradient

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

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

  **Derivative of a logistic function**
  \[
  \frac{\partial}{\partial z_i} \left[ y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \right] = y_i \frac{1}{\mu_i} \frac{\partial g(z_i)}{\partial z_i} + (1 - y_i) \frac{1}{1 - \mu_i} \frac{\partial g(z_i)}{\partial z_i}
  \]
  \[
  = y_i (1 - g(z_i)) + (1 - y_i)(-g(z_i)) = y_i - g(z_i)
  \]

  \[
  \nabla_w l(D, \mathbf{w}) = \sum_{i=1}^{n} -\mathbf{x}_i (y_i - g(\mathbf{w}^T \mathbf{x}_i)) = \sum_{i=1}^{n} -\mathbf{x}_i (y_i - f(\mathbf{w}, \mathbf{x}_i))
  \]
**Limitations of basic linear units**

**Linear regression**
\[ f(x) = w^T x \]

**Logistic regression**
\[ f(x) = p(y=1|x, w) = g(w^T x) \]

- Function linear in inputs!!
- Linear decision boundary!!

---

**Logistic regression. Decision boundary**

Logistic regression model defines a linear decision boundary

- Example: 2 classes (blue and red points)
Linear decision boundary

- Example when logistic regression model is not optimal, but not that bad

When logistic regression fails?

- Example in which the logistic regression model fails
Limitations of logistic regression.

- parity function - no linear decision boundary

Extensions of simple linear units

- use feature (basis) functions to model nonlinearities

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

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

\[\phi_j(x) \quad \text{an arbitrary function of } x\]

The same trick can be done also for the logistic regression
Extension of simple linear units

- **Example:** Fitting of a polynomial of degree \( m \)
  - Data points: pairs of \( <x, y> \)
  - Feature functions:
    \[
    \phi_i(x) = x^i
    \]
  - Function to learn:
    \[
    f(x, w) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i
    \]
  - **On line update** for \( <x, y> \) pair
    \[
    w_0 = w_0 + \alpha(y - f(x, w))
    \]
    \[
    \vdots
    \]
    \[
    w_j = w_j + \alpha(y - f(x, w)) \phi_j(x)
    \]