Linear regression

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

Outline

Linear Regression
- Linear model
- Loss (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. $y_i = f(x_i)$ for all $i = 1, \ldots, n$

- **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

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

  $w_0, w_1, \ldots, w_d$ - **parameters** (weights)

  Bias term $\rightarrow 1$

  Input vector $\langle x_1, x_2, \ldots, x_d \rangle$

  $$ \sum f(x, w) $$
Linear regression

- **Shorter (vector) definition of the model**
  - Include bias constant in the input vector
    \[ \mathbf{x} = (1, x_1, x_2, \cdots, x_d) \]
    \[ f(\mathbf{x}) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d = \mathbf{w}^T \mathbf{x} \]
    \( w_0, w_1, \ldots, w_k \) - parameters (weights)

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

Linear regression. Error.

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

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

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

• 1 dimensional input \( \mathbf{x} = (x_1) \)

\[ \begin{pmatrix} -1.5 \\ -1 \\ -0.5 \\ 0 \\ 0.5 \\ 1 \\ 1.5 \\ 2 \\ 3 \end{pmatrix} \]

\[ \begin{pmatrix} -15 \\ -10 \\ -5 \\ 0 \\ 5 \\ 10 \\ 15 \\ 20 \\ 25 \\ 30 \end{pmatrix} \]

Linear regression. Example.

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

\[ \begin{pmatrix} -3 \\ -2 \\ -1 \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} \]

\[ \begin{pmatrix} -4 \\ -2 \\ 0 \\ 2 \\ 4 \end{pmatrix} \]

\[ \begin{pmatrix} -20 \\ -15 \\ -10 \\ -5 \\ 0 \\ 5 \\ 10 \\ 15 \\ 20 \\ 25 \end{pmatrix} \]
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:
\[ \text{grad}_w (J_n(\mathbf{w})) = \nabla_w (J_n(\mathbf{w})) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i = \mathbf{0} \]
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

\[
Aw = b
\]

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

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

\[
\vdots
\]

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

\[
\vdots
\]

\[
\sum_{i=1}^{n} y_i x_{i,j}
\]

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

\[
Aw = b
\]

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

Solution to SLE: ?
Solving linear regression

• The optimal set of weights satisfies:

\[ \nabla_w (J_n(w)) = \frac{2}{n} \sum_{i=1}^{n} (y_i - w^T x_i) x_i = \mathbf{0} \]

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

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

Solution to SLE:

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

• matrix inversion

Gradient descent solution

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

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

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

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

\[ \alpha > 0 \quad \text{a learning rate} \quad \text{(scales the gradient changes)} \]
Gradient descent method

- Descend using the gradient information

\[
\nabla_w Error(w) |_{w^*}
\]

Direction of the descent

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

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

- New value of the parameter

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

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

- Iteratively approaches the optimum of the Error function

![Gradient descent method](image)

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

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

- vector form:
  \[ w \leftarrow w - \alpha \nabla_w Error_i(w) \]

  $\alpha > 0$ - 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:

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

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

**Fixed learning rate:** \( \alpha(i) = C \)

**Annealed learning rate:** \( \alpha(i) = \frac{1}{i} \)

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

**do**

**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

---

CS 2750 Machine Learning
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
  \[
  \bar{x}_j = \frac{1}{n} \sum_{i=1}^{n} x_{i,j} \quad \quad \sigma_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_{i,j} - \bar{x}_j)^2
  \]

  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** \( \mathbf{y} \)

---

Extensions of simple linear model

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

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

\( \phi_j(\mathbf{x}) \) - an arbitrary function of \( \mathbf{x} \)

The same techniques as before to learn the weights
Additive linear models

- Models linear in the parameters we want to fit
  \[ f(x) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) \]

  \( w_0, w_1 \ldots w_m \) - parameters
  \( \phi_1(x), \phi_2(x) \ldots \phi_m(x) \) - feature or basis functions

- Basis functions examples:
  - a higher order polynomial, one-dimensional input \( x = (x_i) \)
    \[ \phi_1(x) = x \quad \phi_2(x) = x^2 \quad \phi_3(x) = x^3 \]
  - Multidimensional quadratic \( x = (x_1, x_2) \)
    \[ \phi_1(x) = x_1 \quad \phi_2(x) = x_1^2 \quad \phi_3(x) = x_2 \quad \phi_4(x) = x_2^2 \quad \phi_5(x) = x_1x_2 \]
  - Other types of basis functions
    \[ \phi_1(x) = \sin x \quad \phi_2(x) = \cos x \]

Fitting additive linear models

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

Assume:
\[ \Phi(x_i) = (1, \phi_1(x_i), \phi_2(x_i), \ldots, \phi_m(x_i)) \]

\[ \nabla_w J_n(w) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - f(x_i)) \Phi(x_i) = \mathbf{0} \]

- Leads to a system of \( m \) linear equations

\[ w_0 \sum_{i=1}^{n} \phi_j(x_i) + \ldots + w_j \sum_{i=1}^{n} \phi_j(x_i) \phi_j(x_i) + \ldots + w_m \sum_{i=1}^{n} \phi_m(x_i) \phi_j(x_i) = \sum_{i=1}^{n} y_i \phi_j(x_i) \]

- Can be solved exactly like the linear case
Example. Regression with polynomials.

Regression with polynomials of degree $m$

- **Data points:** pairs of $<x, y>$
- **Feature functions:** $m$ feature functions
  \[
  \phi_i(x) = x^i \quad i = 1, 2, \ldots, m
  \]
- **Function to learn:**
  \[
  f(x, w) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i
  \]

Learning with feature functions

**Function to learn:**
\[
 f(x, w) = w_0 + \sum_{i=1}^{k} w_i \phi_i(x)
\]

**On line gradient update** for the $<x, y>$ pair
\[
 w_0 = w_0 + \alpha (y - f(x, w)) \\
 \vdots \\
 w_j = w_j + \alpha (y - f(x, w)) \phi_j(x)
\]

Gradient updates are of the same form as in the linear and logistic regression models.
Example. Regression with polynomials.

**Example**: Regression with polynomials of degree $m$

\[ f(x, w) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i \]

- **On line update** for \(<x,y>\) pair

\[
\begin{align*}
    w_0 &= w_0 + \alpha (y - f(x, w)) \\
    &\vdots \\
    w_j &= w_j + \alpha (y - f(x, w)) x^i
\end{align*}
\]
Multidimensional additive model example

Statistical model of regression

- **A generative model:** \( y = f(x, \omega) + \epsilon \)
  
  \( f(x, \omega) \) is a deterministic function
  
  \( \epsilon \) is a random noise, represents things we cannot capture with \( f(x, \omega) \), e.g. \( \epsilon \sim N(0, \sigma^2) \)

Assume \( f(x, \omega) = \omega^T x \) is a linear model, and \( \epsilon \sim N(0, \sigma^2) \)

Then: \( f(x, \omega) = E(y \mid x) \) models the mean of outputs \( y \) for \( x \) and the **noise** models deviations from the mean

- **The model defines the conditional density** of \( y \) given \( x, \omega, \sigma \)

\[
p(y \mid x, \omega, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} \exp \left[ -\frac{1}{2\sigma^2} (y - f(x, \omega))^2 \right]
\]
ML estimation of the parameters

- **likelihood of predictions** = the probability of observing outputs \( y \) in \( D \) given \( w, \sigma \)
  \[
  L(D, w, \sigma) = \prod_{i=1}^{n} p(y_i \mid x_i, w, \sigma)
  \]

- **Maximum likelihood estimation of parameters**
  - parameters maximizing the likelihood of predictions
  \[
  w^* = \arg \max_w \prod_{i=1}^{n} p(y_i \mid x_i, w, \sigma)
  \]

- **Log-likelihood** trick for the ML optimization
  - Maximizing the log-likelihood is equivalent to maximizing the likelihood
  \[
  l(D, w, \sigma) = \log(L(D, w, \sigma)) = \log \prod_{i=1}^{n} p(y_i \mid x_i, w, \sigma)
  \]

ML estimation of the parameters

- **Using conditional density**
  \[
  p(y \mid x, w, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} \exp[-\frac{1}{2\sigma^2}(y - f(x, w))^2]
  \]

- **We can rewrite the log-likelihood as**
  \[
  l(D, w, \sigma) = \log(L(D, w, \sigma)) = \log \prod_{i=1}^{n} p(y_i \mid x_i, w, \sigma)
  = \sum_{i=1}^{n} \log p(y_i \mid x_i, w, \sigma)
  = \sum_{i=1}^{n} \left\{ -\frac{1}{2\sigma^2}(y_i - f(x_i, w))^2 - c(\sigma) \right\}
  = -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - f(x_i, w))^2 + C(\sigma)
  \]

- Maximizing with regard to \( w \), is **equivalent to minimizing squared error functions**
ML estimation of parameters

- Criteria based on mean squares error function and the log likelihood of the output are related
  \[ J_{\text{online}}(y_i, x_i) = \frac{1}{2\sigma^2} \log p(y_i \mid x_i, w, \sigma) + c(\sigma) \]

- We know how to optimize parameters \( w \)
  – the same approach as used for the least squares fit
- But what is the ML estimate of the variance of the noise?
- Maximize \( l(D, w, \sigma) \) with respect to variance
  \[ \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i, w^*))^2 \]
  \[ = \text{mean square prediction error for the best predictor} \]

Regularized linear regression

- If the number of parameters is large relative to the number of data points used to train the model, we face the threat of overfit (generalization error of the model goes up)
- The prediction accuracy can be often improved by setting some coefficients to zero
  – Increases the bias, reduces the variance of estimates
- Solutions:
  – Subset selection
  – Ridge regression
  – Lasso regression
  – Principal component regression

- Next: ridge regression
Ridge regression

• Error function for the standard least squares estimates:
  \[ J_n(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 \]
• We seek: \( w^* = \arg\min_w \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 \)
• Ridge regression:
  \[ J_n(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|^2 \]
• Where \( \|w\|^2 = \sum_{i=0}^{d} w_i^2 \) and \( \lambda \geq 0 \)
• What does the new error function do?

Ridge regression

• Standard regression:
  \[ J_n(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 \]
• Ridge regression:
  \[ J_n(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_{L_2}^2 \]
• \( \|w\|_{L_2}^2 = \sum_{i=0}^{d} w_i^2 \) penalizes non-zero weights with the cost proportional to \( \lambda \) (a shrinkage coefficient)
• If an input attribute \( x_j \) has a small effect on improving the error function it is “shut down” by the penalty term
• Inclusion of a shrinkage penalty is often referred to as regularization
Regularized linear regression

How to solve the least squares problem if the error function is enriched by the regularization term $\lambda \|w\|^2$?

**Answer:** The solution to the optimal set of weights $w$ is obtained again by solving a set of linear equation.

**Standard linear regression:**

$$\nabla_w (J_n(w)) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w^T x_i) x_i = 0$$

**Solution:** $w^* = (X^T X)^{-1} X^T y$

where $X$ is an $n \times d$ matrix with rows corresponding to examples and columns to inputs

**Regularized linear regression:**

$$w^* = (\lambda I + X^T X)^{-1} X^T y$$

---

Lasso regression

- **Standard regression:**

$$J_n(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

- **Lasso regression/regularization:**

$$J_n(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_{L_1}$$

- $\|w\|_{L_1} = \sum_{i=0}^{d} |w_i|$ penalizes non-zero weights with the cost proportional to $\lambda$.

- L1 is more aggressive pushing the weights to 0 compared to L2.