Lecture 9: Linear Models + Logistic Regression

Alaa Tharwat
Fachhochschule Bielefeld
120 PUBLICATIONS **6,195** CITATIONS

Lecture 9: Linear Model

Alaa Tharwat

Lecture 9: Linear Model

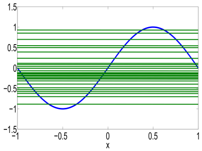
- Review of Lecture 8
- Nonlinear transformation
- Logistic regression

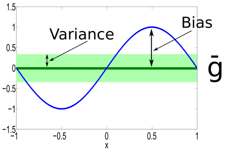
Lecture 9: Linear Model

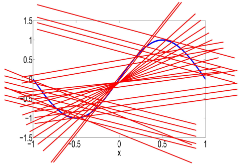
- Review of Lecture 8
- Nonlinear transformation
- Logistic regression

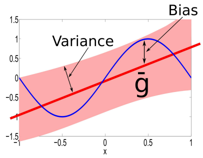
$$E_D[E_{in}(g^{(D)}(x))] = E_D[g^{(D)}(x) - f(x))^2] + \overset{\text{Irreducible error}}{\boxed{E_D[\epsilon^2]}}$$

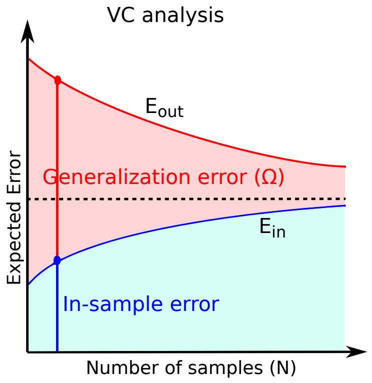
$$E_D[(g^{(D)}(x) - f(x))^2] = \overset{\text{Variance}}{\boxed{E_D[(g^{(D)}(x) - \bar{g}(x))^2]}} + \overset{\text{Bias}}{\boxed{(\bar{g}(x) - f(x))^2}}$$

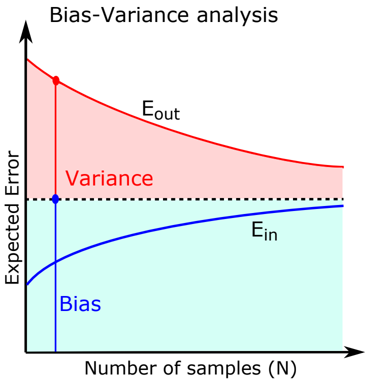


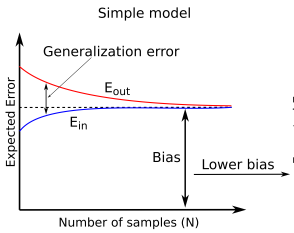


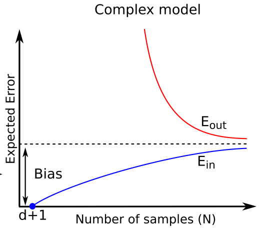





VC analysis
Expected Error
E_{out}
Generalization error (Ω)
E_{in}
In-sample error
Number of samples (N)


Bias-Variance analysis
Expected Error
E_{out}
Variance
E_{in}
Bias
Number of samples (N)

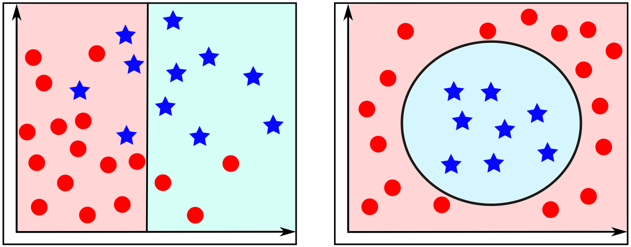

Simple model
Generalization error
Expected Error
E_{out}
E_{in}
Bias
Lower bias
Number of samples (N)


Complex model
Expected Error
E_{out}
E_{in}
Bias
d+1
Number of samples (N)

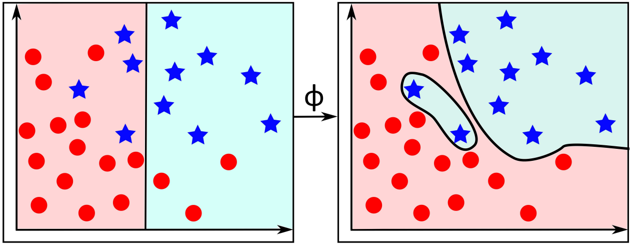
Lecture 9: Linear Model

- Review of Lecture 8
- Nonlinear transformation
- Logistic regression

$$\begin{tikzcd}
\mathbf{x} = (x_0, x_1, \dots, x_d) \arrow[r, "\phi"] \arrow[d] & \mathbf{z} = (z_0, z_1, \dots, z_{\tilde{d}}) \arrow[d] \\
\mathbf{w} & \tilde{\mathbf{w}} \\
d_{VC} = d + 1 & d_{VC} \leq \tilde{d} + 1
\end{tikzcd}$$



- We can accept with in-sample error ($E_{in} > 0$), or,
- Transform the data into higher dimensional space to get $E_{in} = 0$, is this solution is good/suitable?
  - Answer: remember that the goal of any learning algorithm is to learn the model to estimate unseen data, and reducing E_{in} is not an indicator for a good model because as we discussed before this may increase the model complexity and hence reduce the bias but increase the variance (this problem is called the overfitting problem)

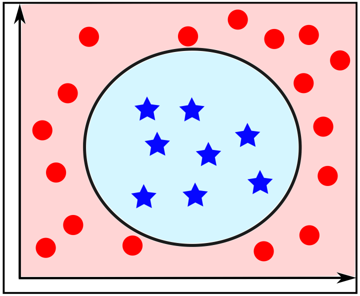


- Transform data into higher-dimensional spaces increases the VC dimension
- The problem of generalization in higher dimensional space is sometimes balanced by the advantage we get in approximating the target function better. Hence, we cannot avoid the approximation-generalization tradeoff

Nonlinear transformation

- $d_{VC} \uparrow$ Higher $\tilde{d}$: better chance of being linearly separable $(E_{in} \downarrow)$

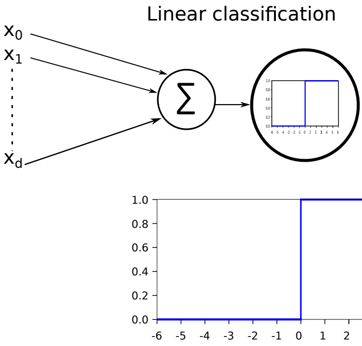
- We can transform the data into higher dimensional space, $\mathbf{z} = \{1, x_1, x_2, x_1x_2, x_1^2, x_2^2\}$
- But, simply we can use, $\mathbf{z} = \{1, x_1^2, x_2^2\}$
- Or, better to reduce it to, $\mathbf{z} = \{1, x_1^2 + x_2^2\}$
- More simpler, $\mathbf{z} = \{x_1^2 + x_2^2 - 0.6\}$
- This is because we can see the data before selecting the model, is this helpful/useful?
    - Answer: No. This will reduce the E_{out}

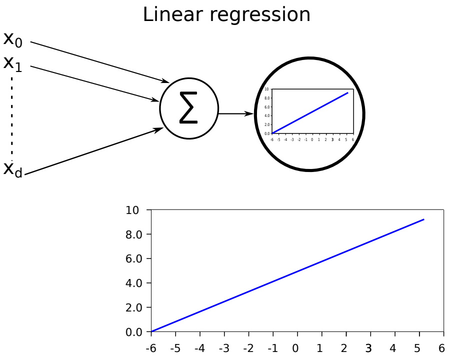


Lecture 9: Linear Model

- Review of Lecture 8
- Nonlinear transformation
- Logistic regression

- In linear classification: $h(x) = \text{sign}(s) = \text{sign}(w_0x_0 + w_1x_1 + \ldots, w_{d+1}x_{d+1}) = \begin{cases} 0 & \text{if} \sum_i w_ix_i \leq 0 \\ 1 & \text{if} \sum_i w_ix_i > 0 \end{cases}$. Here we used, the **step** activation function
  - In the step function, the output is zero when $s < 0$ or one when $s > 0$
- In linear regression: $h(x) = \text{sign}(s) = \text{sign}(w_0x_0 + w_1x_1 + \ldots, w_{d+1}x_{d+1})$. Here we used, **linear** activation function

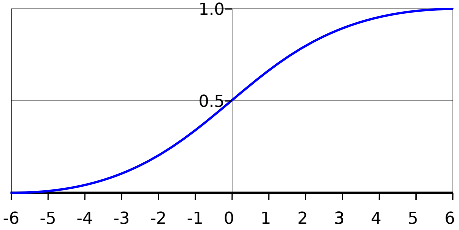

Linear classification
x_0
x_1
x_d
$\sum$
1.0
0.8
0.6
0.4
0.2
0.0
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6


Linear regression
x_0
x_1
x_d
$\sum$
10
8.0
6.0
4.0
2.0
0.0
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6

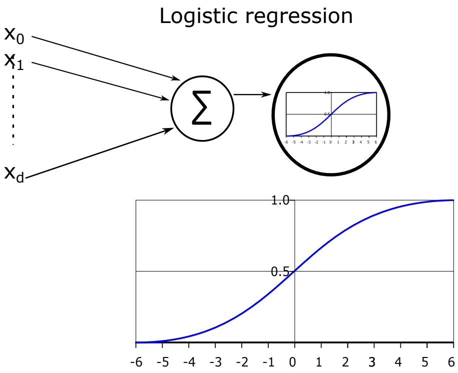
- There is another function is called **Sigmoid** or **soft threshold** function

$$h(x) = \theta(s) = \frac{e^s}{e^s + 1} = \frac{1}{1 + e^{-s}} = \frac{1}{1 + e^{-(w.x)}}$$

- Sigmoid function is similar to the step function, it has the same behavior for large positive and negative numbers, but there is a difference when the input is of modest size
- We can consider that the Sigmoid function is a smooth version from the step function and hence a small change in inputs will produce a small change in the output



- $h(x) = \theta(s)$ is interpreted as a probability, because the output is $\in [0, 1]$.
- For example, assume the input **x** is a set of features such as age, weight, cholesterol level, ... The output $\theta(s)$ is the probability of a heart attack



- Do you remember, $P(y|\mathbf{x})$ which is the target distribution and y is generated by $f \Rightarrow f(\mathbf{x}) = P[y = +1|\mathbf{x}]$. This does not give the value of f explicitly, instead, it gives the probability (i.e. patients who had heart attack and patients who had not)

$$P(y|\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1 \\ 1 - f(\mathbf{x}) & \text{for } y = -1 \end{cases}$$

- The target function $f \in [0, 1]$ (probability)
- $g(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x}) \approx f(\mathbf{x})$
- If $h = f$ (how likely to get y from $\mathbf{x}$?)

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1 \\ 1 - h(\mathbf{x}) & \text{for } y = -1 \end{cases}$$

- In Sigmoid function, $\theta(-s) = 1 - \theta(s)$ and $h(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x})$

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x}) & \text{for } y = +1 \\ 1 - h(\mathbf{x}) = 1 - \theta(\mathbf{w}^T\mathbf{x}) & \text{for } y = -1 \end{cases}$$

- $P(y|\mathbf{x}) = \theta(y\mathbf{w}^T\mathbf{x}) \begin{cases} \theta(\mathbf{w}^T\mathbf{x}) & \text{for } y = +1 \\ \theta(-\mathbf{w}^T\mathbf{x}) = 1 - \theta(\mathbf{w}^T\mathbf{x}) & \text{for } y = -1 \end{cases}$
- The probability of getting the $y_1, \ldots, y_N$ in D from the corresponding $\mathbf{x}_1, \ldots, \mathbf{x}_N$ is $P(y_1, \ldots, y_n|\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{n=1}^{N} P(y_n|\mathbf{x}_n) = \prod_{i=1}^{N} \theta(y_n\mathbf{w}^T\mathbf{x}_n)$

- To maximize the likelihood, select h that maximizes the probability

$$\min \; -\frac{1}{N} ln(\prod_{n=1}^{N} \theta(y_n \mathbf{w}^T \mathbf{x}_n)) = \frac{1}{N} \sum_{n=1}^{N} ln(\frac{1}{\theta(y_n \mathbf{w}^T \mathbf{x}_n)})$$

$$\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}} \Rightarrow ln(\frac{1}{\theta(s)}) = ln(\frac{1}{\frac{1}{1+e^{-s}}}) = ln(1+e^{-s})$$

$$E_{in} = \frac{1}{N} \sum_{n=1}^{N} ln(1+e^{-y_n \mathbf{w}^T \mathbf{x}_n}) \rightarrow \textbf{Cross-entropy error}$$

- if the sample is correctly classified $\mathbf{w}^T \mathbf{x}_n$ and y_n have the same sign $\Rightarrow e^{-\text{value}} \Rightarrow$ small value
- if the sample is misclassified $\mathbf{w}^T \mathbf{x}_n$ and y_n have different signs $\Rightarrow e^{+\text{value}} \Rightarrow$ large value

- In linear regression, the solution is a closed-form solution

$$E_{in} = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{w}^T \mathbf{x}_n - y_n)^2$$

- In logistic regression, the solution is iterative[1]

$$E_{in} = \frac{1}{N} \sum_{n=1}^{N} ln(1 + e^{-y_n \mathbf{w}^T \mathbf{x}_n})$$

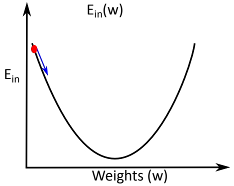
- When $y_n \mathbf{w}^T \mathbf{x}_n$ is large and positive the error will be small

[1] Closed-form expressions are generally easier to work with than iterative equations (where we cannot find a formula), but it is not always (or even often) possible to come up with a closed-form way of expressing a given process. So, iterative expressions have the great advantage of being extremely flexible.

- The first step is to initialize $\mathbf{w}(0)$, next, update/adjust the weight
- This is suitable for "twice-differentiable" functions[2]

$$\mathbf{w}(1) = \mathbf{w}(0) + \eta \hat{v}$$

where $\hat{v}$ is the direction of a unit vector, we pick $\hat{v}$ to make $E_{in}(w(t+1))$ as small as possible.



$$\begin{aligned}\Delta E_{in} &= E_{in}(\mathbf{w}(1)) - E_{in}(\mathbf{w}(0)) \\ &= E_{in}(\mathbf{w}(0) + \eta \hat{v}) - E_{in}(\mathbf{w}(0))\end{aligned}$$

[2]The function that its second derivative is defined for all input values within its range.

Since η is small, using Taylor expansion[3] to the first order, the change in E_{in} is

$$\begin{aligned}\Delta E_{in} &= E_{in}(\mathbf{w}(0) + \eta\hat{v}) - E_{in}(\mathbf{w}(0)) \\ &= \eta \bigtriangledown E_{in}(\mathbf{w}(0)^T\hat{v}) + \overset{\text{small}}{\boxed{O(\eta^2)}} \\ &\geq \eta|| \bigtriangledown E_{in}(\mathbf{w}(0))||\end{aligned}$$

This is the direction leads to the largest decrease in E_{in} for a given η

[3] Taylor series: $f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)(x-a)^n}{n!}$. For example, $f(x) = e^x$; hence, $f(x) = f'(x) = f''(x) = f''(x)$ and $f(0) = f'(0) = f''(0) = f'''(0) = 1$. $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$.

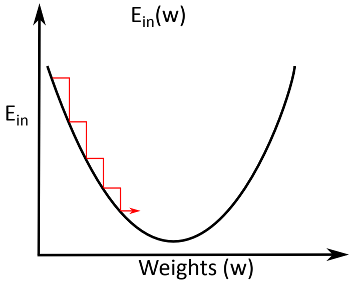
$$\hat{v} = -\frac{\bigtriangledown E_{in}(\mathbf{w}(0))}{|| \bigtriangledown E_{in}(\mathbf{w}(0))||}$$

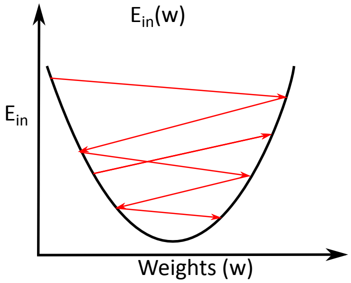
$$\mathbf{w}(1) = \mathbf{w}(0) + \eta\hat{v} = \mathbf{w}(0) - \eta\frac{\bigtriangledown E_{in}(\mathbf{w}(0))}{\|\bigtriangledown E_{in}(\mathbf{w}(0))\|} = \mathbf{w}(0) - \eta \bigtriangledown E_{in}(\mathbf{w}(0))$$

- $\Delta\mathbf{w} = \eta\hat{v} = -\eta\frac{\bigtriangledown E_{in}(\mathbf{w}(0))}{\|\bigtriangledown E_{in}(\mathbf{w}(0))\|} \Rightarrow \Delta\mathbf{w} = -\eta \bigtriangledown E_{in}(\mathbf{w}(0))$
- We need to minimize E_{in}
- η is the learning rate not a fixed step

How η affects the algorithm?

- Small η, the algorithm will be slow
- Large η, the algorithm will be fast, but maybe not converge
- It is better to have variable η increases with the slope





Algorithm 1 : Logistic regression algorithm

```
1: Initialize the weights $w(0)$
2: for all $t = 0, 1, \dots$ do
3:     Compute the gradient
           $$\bigtriangledown E_{in} = -\frac{1}{N}\sum_{i=1}^{N}\frac{y_i\mathbf{x}_i}{1+e^{y_i\mathbf{w}^T(t)\mathbf{x}_i}}$$
4:     Update weights: $\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \bigtriangledown E_{in}$
5:     Iterate until the model converges to a small error
6: end for
7: Return the final weight $\mathbf{w}$
```

- Termination criteria
  - an upper bound on the number of iterations
  - the change in the gradient is almost zero

Before a weight update is done

- **Batch gradient descent**: The gradient is computed for the error on the whole data
- **Stochastic gradient descent (SGD)**: The gradient is computed for the error on each training sample. That is "on average" the minimization proceeds in the right direction (as in the batch gradient), but randomly.
- The computational cost of the SGD algorithm is cheaper
- The randomness in SGD helps SGD to avoid the local minima problem

In the credit approval problem

- Classification:
  - Perceptron (PLA or Pocket): we used the classification error and the output is the class label (approve/deny)

$$E_{in} = \begin{cases} \sum_{i=0}^{d} w_i x_i \geq \text{threshold} & \text{Approve credit} \\ \sum_{i=0}^{d} w_i x_i < \text{threshold} & \text{Deny credit} \end{cases} \tag{1}$$

  - Logistic regression (Gradient descent): we used the cross-entropy error and the output is the probability of default

$$E_{in} = \frac{1}{N} \sum_{n=1}^{N} ln(1 + e^{-y_n \mathbf{w}^T \mathbf{x}_n})$$

- Regression: Linear regression (Pseudo-inverse) we used the squared error and the output is the amount of credit (in dollars)

$$E_{in} = \frac{1}{N} \sum_{i=1}^{N} (h(\mathbf{x}_i) - y_i)^2$$