Logistic Regression

Alla Rozovskaya

Slides were created by Alla Rozovskaya and are based on Chapter 5 of “Speech and Language Processing” by Jurafsky and Martin, 3rd edition.

Logistic Regression

- A supervised machine learning algorithm for classification
- LR has a close relationship to neural networks
- LR can be used as a binary classifier or can be extended to multi-class classification

Components of a probabilistic machine learning classifier

(1) A feature representation of the input, i.e. **feature vector**

(2) (2) **A classification function** that computes **the estimated class** $\hat{y}$, via $p(y|x)$:

sigmoid

softmax

(3) An objective function for learning (usually involves **minimizing training error)**

(4) An algorithm for optimizing the objective function

e.g. Stochastic gradient descent

Two phases of a supervised machine learning algorithm (for LR)

- **Training**: we train the system (specifically the weights *w* and *b*) *using stochastic*

gradient descent and the cross-entropy loss.

- **Test:** Given a test example x we compute p(y|x) and return the higher probability label y = 1 or y = 0.

The sigmoid function

- Let a single input observation *x* be represented by a **vector of features** $[x_1, x_2, \ldots x_n]$
- The classifier outputs 1 or 0
- We want to know **the probability** $P(y = 1|x)$ that this observation is a member of the class, for example (for sentiment classification):
  - $P(y = 1|x)$ is the probability that the document has positive sentiment, while and $P(y = 0|x)$ is the probability that the document has negative sentiment.

Learning in Logistic Regression

- Logistic regression learns, **from a training set**:
  - a **vector of weights** and a **bias term**
- Each weight w_i is a real number, and is associated with one of the input features x_i
- The weight w_i represents how important that input feature is to the classification decision
- w_i can be positive (meaning the feature is associated with the class) or negative (meaning the feature is not associated with the class).
  - Thus we might expect in a sentiment task the word **awesome** to have a high positive weight, and **abysmal** bias term to have a very negative weight.
- The **bias term** is another real number that’s added to the weighted input

Making decision on a test instance

- After we've learned the weights in training
- The classifier first multiplies each x_i by its weight w_i, sums up the weighted features, and adds the bias term b.
- z expresses the weighted sum of the evidence for the class:

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

Dot product notation

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

$$z = w \cdot x + b$$

Nothing forces z to lie between 0 and 1. z ranges from $-\infty$ to ∞

The sigmoid function

- To create a probability, we'll pass z through **sigmoid function** σ(z): $y = \sigma(z) = \frac{1}{1+e^{-z}}$

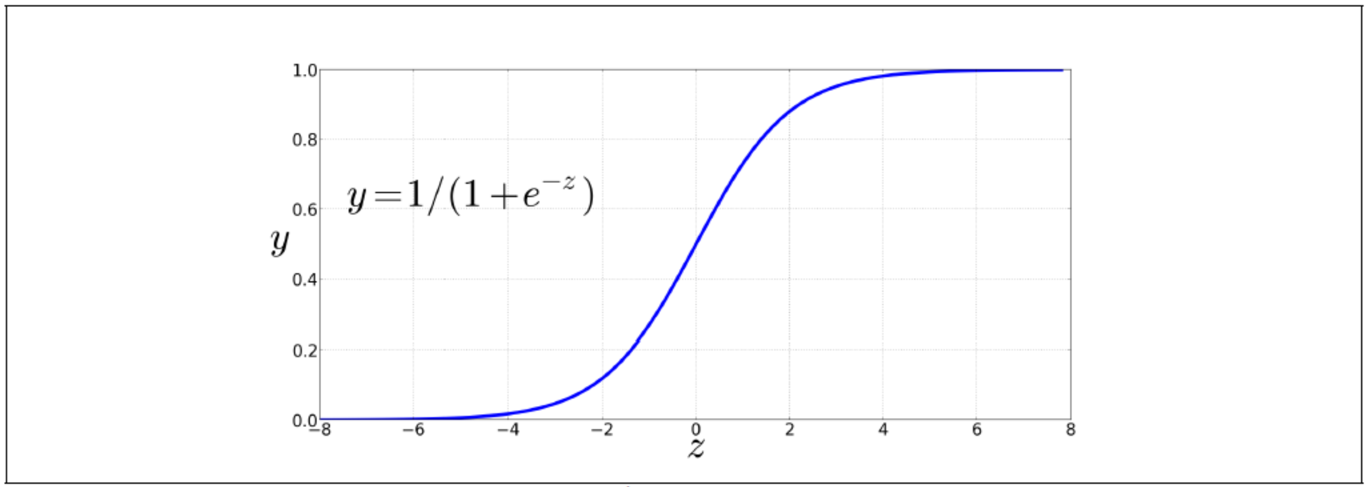


Figure 5.1 The sigmoid function $y = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range [0, 1]. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

The sigmoid function

- The sigmoid function takes a real-valued number and maps it into the **range between 0 and 1**
- And it's **differentiable**, which is helpful for learning.

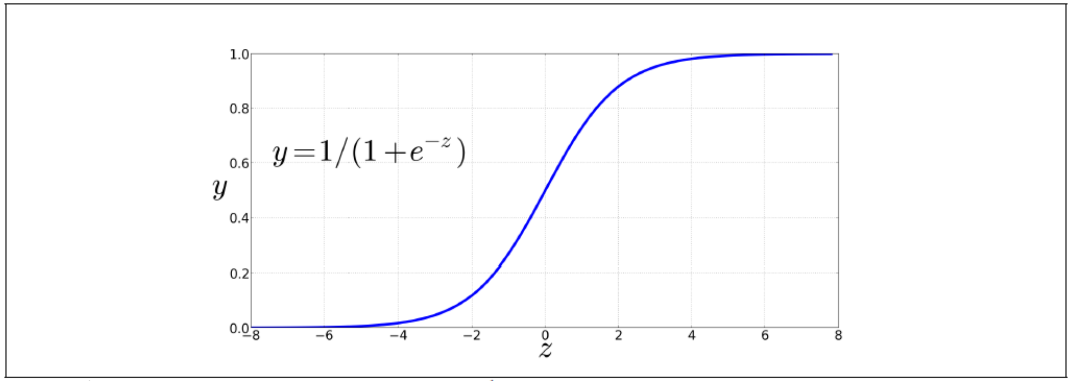


Figure 5.1 The sigmoid function $y = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $[0,1]$. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

Mapping dot products to probabilities

- The sigmoid maps the dot product into the range between 0 and 1
- To make sure these correspond to probabilities, we make sure they sum up to 1:

$$
\begin{aligned}
P(y=1) &= \sigma(w \cdot x + b) \\
&= \frac{1}{1+e^{-(w \cdot x + b)}}
\end{aligned}
$$

$$
\begin{aligned}
P(y=0) &= 1 - \sigma(w \cdot x + b) \\
&= 1 - \frac{1}{1+e^{-(w \cdot x + b)}} \\
&= \frac{e^{-(w \cdot x + b)}}{1+e^{-(w \cdot x + b)}}
\end{aligned}
$$

Making decision on a test instance

- Now we have an algorithm that given an instance x computes the probability

  P(y =1|x).
- For a test instance x, we say yes if the probability P(y = 1|x) is more than .5:

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Example: sentiment classification on movie reviews

- unbelievably disappointing

- Full of zany characters and richly applied satire, and some great plot twists

- this is the greatest screwball comedy ever filmed

- It was pathetic. The worst part about it was the boxing scenes.

Sentiment classification: features

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Weights: $[2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$, while $b = 0.1$.

Sentiment classification

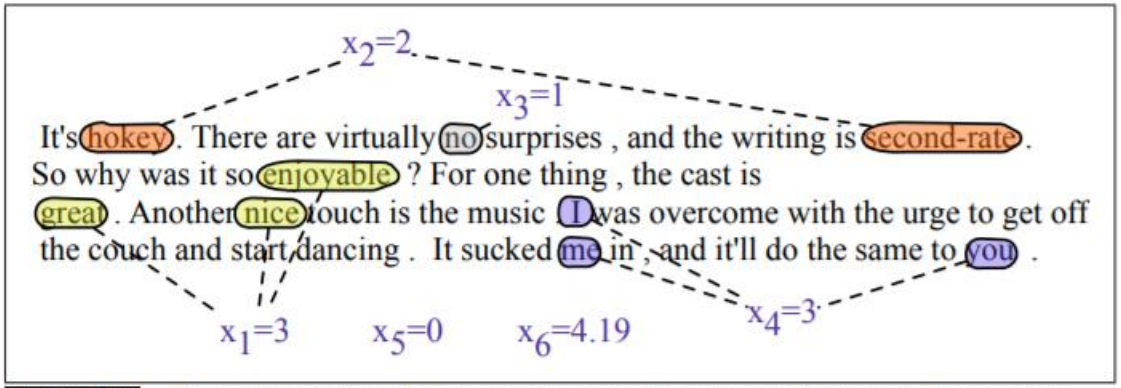


Figure 5.2 A sample mini test document showing the extracted features in the vector x.

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Computing P(+|x) and P(-|x)

$$\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\
&= \sigma(.833) \\
&= 0.70 \qquad (5.6) \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.30
\end{aligned}$$

LR for period disambiguation

$$x_1 = \begin{cases} 1 & \text{if ``}Case(w_i) = \text{Lower''} \\ 0 & \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1 & \text{if ``}w_i \in \text{AcronymDict''} \\ 0 & \text{otherwise} \end{cases}$$

$$x_3 = \begin{cases} 1 & \text{if ``}w_i = \text{St. \& } Case(w_{i-1}) = \text{Cap''} \\ 0 & \text{otherwise} \end{cases}$$

Learning in LR

- What the system produces via Eq. 5.5 is ŷ, the **system's estimate of the true y:**

$$
\begin{aligned}
P(y=1) &= \sigma(w \cdot x + b) \\
&= \frac{1}{1+e^{-(w \cdot x + b)}} \\
P(y=0) &= 1 - \sigma(w \cdot x + b) \\
&= 1 - \frac{1}{1+e^{-(w \cdot x + b)}} \\
&= \frac{e^{-(w \cdot x + b)}}{1+e^{-(w \cdot x + b)}}
\end{aligned}
$$

We want to learn **parameters** (w and b) that make ŷ for each training observation as close as possible to the true y

Learning in LR: loss function and updating the weights

- Loss (cost) function: a metric for measuring **the distance** between the system output and the gold output
- The second thing we need is an **optimization algorithm for iteratively updating the weights** so as to minimize this loss function.

The loss function: cross-entropy loss

$$L(\hat{y}, y) = \text{How much } \hat{y} \text{ differs from the true } y$$

We do this via a loss function that **prefers the correct class labels of the training examples to be more likely**.

We choose the **parameters w,b** that **maximize the log probability of the true y labels in the training data** given the observations x.

Cross-entropy loss

- The equation simplifies to ŷ if y=1; if y=0, the equation simplifies to (1− ŷ):

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

- Take the log of both sides:

$$\begin{aligned}\log p(y|x) &= \log\left[\hat{y}^y (1-\hat{y})^{1-y}\right] \\ &= y\log\hat{y} + (1-y)\log(1-\hat{y})\end{aligned}$$

Cross-entropy loss

- The equation below is log likelihood (so should be maximized:

$$\begin{aligned}\log p(y|x) &= \log\left[\hat{y}^{y}(1-\hat{y})^{1-y}\right] \\ &= y\log\hat{y} + (1-y)\log(1-\hat{y})\end{aligned}$$

- We can turn it into loss function by flipping the sign

$$L_{CE}(\hat{y},y) = -\log p(y|x) \;=\; -[y\log\hat{y} + (1-y)\log(1-\hat{y})]$$

Cross-entropy loss

$$L_{CE}(\hat{y}, y) = -\log p(y|x) \;=\; -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$$

Finally, we can plug in the definition of $\hat{y} = \sigma(w \cdot x + b)$:

$$L_{CE}(w,b) \;=\; -[y \log \sigma(w \cdot x + b) + (1-y) \log(1 - \sigma(w \cdot x + b))]$$

Loss function example

- Suppose y=1

$$
\begin{aligned}
L_{CE}(w,b) = & -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))] \\
= & -[\log\sigma(w\cdot x+b)] \\
= & -\log(.69) \\
= & .37
\end{aligned}
$$

Loss function example

- Suppose y=0

$$
\begin{aligned}
L_{CE}(w,b) &= -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))] \\
&= -[\log(1-\sigma(w\cdot x+b))] \\
&= -\log(.31) \\
&= 1.17
\end{aligned}
$$

Minimizing the loss function

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(y^{(i)}, x^{(i)}; \theta)$$

Loss function for LR

- The loss function in LR is convex – one global minimum (note that the loss function below is NOT convex!

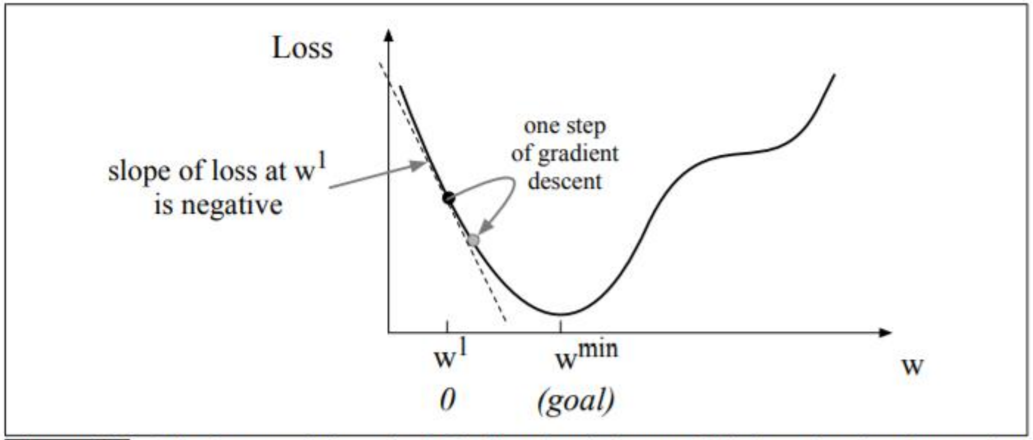
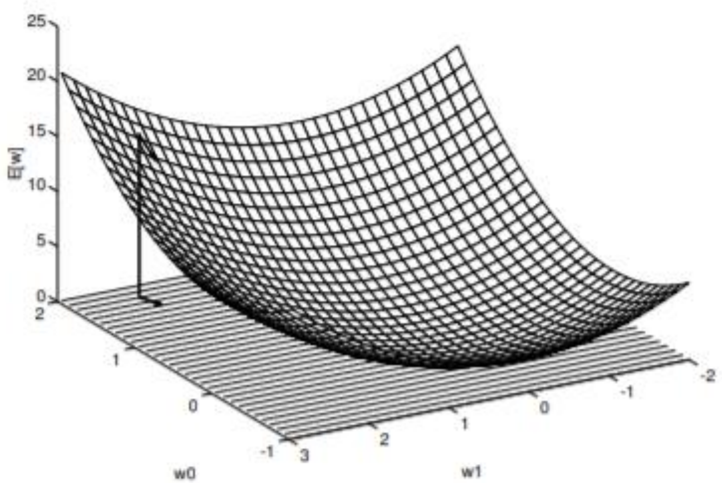


Figure 5.3 The first step in iteratively finding the minimum of this loss function, by moving w in the reverse direction from the slope of the function. Since the slope is negative, we need to move w in a positive direction, to the right. Here superscripts are used for learning steps, so w^1 means the initial value of w (which is 0), w^2 at the second step, and so on.

Visualizing the hypothesis space



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

For a linear unit with 2 weights, the hypothesis space is a w_0, w_1 hyperplane.

The vertical axis indicates the error of the corresponding weight vector hypothesis, relative to a fixed set of training examples – **we desire the minimum error**

Given our error definition, for linear units the error surface is a parabola with a single global minimum.

The arrow shows the negated gradient at one particular point, indicating the direction in the w_0, w_1 plane producing steepest descent along the error surface.

Derivation of the Gradient Descent rule

- How can we calculate the direction of the steepest descent along the error surface?
- This can be found by **computing the derivative of the loss function w.r.t. each component of vector w**.
- The vector derivative is called **the gradient with respect to w**.

The gradient of L with respect to w

$$\nabla_{\theta} L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$

The gradient is itself **a vector, whose components are partial derivatives of L with respect to each w_i**. The gradient specifies the direction that produces the steepest increase in L. **The negative of this vector** therefore gives the direction of steepest decrease.

The gradient for Logistic Regression

- Cross-entropy loss

$$L_{CE}(w,b) = -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))]$$

- The derivative of this function for one observation is as follows (see text for more detail):

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w\cdot x+b)-y]x_j$$

Note that the gradient with respect to a single weight w_j represents a very intuitive value: **the difference between the true y and our estimated ŷ = σ(w· x+b) for that observation, multiplied by the corresponding input value** x_j .

The stochastic gradient descent algorithm

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ
    # where: L is the loss function
    #   f is a function parameterized by θ
    #   x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$
    #   y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$

$\theta \leftarrow 0$
repeat til done   # see caption
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
    1. Optional (for reporting):   # How are we doing on this tuple?
        Compute $\hat{y}^{(i)} = f(x^{(i)};\theta)$ # What is our estimated output $\hat{y}$?
        Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?
    2. $g \leftarrow \nabla_{\theta} L(f(x^{(i)};\theta), y^{(i)})$ # How should we move θ to maximize loss?
    3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead
return θ

Figure 5.5 The stochastic gradient descent algorithm. Step 1 (computing the loss) is used to report how well we are doing on the current tuple. The algorithm can terminate when it converges (or when the gradient $< \epsilon$), or when progress halts (for example when the loss starts going up on a held-out set).

Mini-batch training

- **Stochastic gradient descent** is called stochastic because it chooses a single random example at a time
- In **batch training** we compute the gradient over the entire dataset
- **Mini-batch training**: we train on a group of m examples (perhaps 512, or 1024) that is less than the whole dataset

Cost function for mini-batch GD

$$
\begin{aligned}
Cost(w,b) &= \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)}) \\
&= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \sigma(w\cdot x^{(i)} + b) + (1-y^{(i)}) \log \left(1-\sigma(w\cdot x^{(i)}+b)\right)
\end{aligned}
$$

It's the average loss over m examples

$$
\frac{\partial Cost(w,b)}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m} \left[\sigma(w\cdot x^{(i)} + b) - y^{(i)}\right] x_j^{(i)}
$$

The mini-batch gradient is the average of the gradients