Logistic Regression

A Discriminative Classifier

Generative and Discriminative Classifiers

- The naive Bayes assigns a class c to a document d ***not by directly computing P(c|d)*** but by computing a likelihood and a prior.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\ P(d|c)\ P(c)$$

likelihood of the document

prior probability of the class

- **A generative model (**like naive Bayes) makes use of *likelihood term*, which expresses *how to generate the features of a document if we knew it was of class c*.

- A **discriminative model** (like logistic regression) in the text categorization scenario *attempts model to directly compute P(c|d).*
  - It will learn to assign high weight to document features that directly improve its ability to discriminate between possible classes, even if it couldn't generate an example of one of the classes.

Generative and Discriminative Classifiers

- **Discriminative (conditional) models** are widely used in NLP (and ML generally).
  - They give high accuracy performance
  - They make it easy to incorporate lots of linguistically important features
  - They allow automatic building of language independent NLP modules

- In text classification task, we have some data {(d, c)} of paired observations d and hidden classes c.

- **Generative (joint) models** place probabilities over both observed data and the hidden stuff (generate the observed data from hidden stuff).
  - *some generative models:* n-gram models, Naive Bayes classifiers, hidden Markov models, probabilistic context-free grammars, …

- **Discriminative (conditional) models** take the data as given, and put a probability over hidden structure given the data.
  - *some discriminative models:* logistic regression, maximum entropy models, conditional random fields, SVMs, perceptron, …

Components of
A Probabilistic Machine Learning Classifier

- Machine learning classifiers require a training corpus of observations input/output pairs (x,y).
  - In text classification, document/class (d,c) pairs.

- A machine learning system for classification has four components:

  1. **A feature representation of the input**. For each input observation x, this will be a vector of features $[x_1,x_2,\ldots,x_n]$.
  2. **A classification function that computes $\hat{y}$ (the estimated class)** via P(y|x). We will use ***sigmoid function*** in logistic regression.
  3. **An objective function for learning**, usually involving minimizing error on training examples. We will use ***cross-entropy loss function*** in logistic regression.
  4. **An algorithm for optimizing the objective function**. We will use ***stochastic gradient descent algorithm*** in logistic regression.

Logistic Regression

- The goal of **binary logistic regression** is to train a classifier that can make a binary decision about the class of a new input observation.

- **Training:** we train the system to learn a vector of weights for features and a bias term using ***stochastic gradient descent*** and ***cross-entropy loss function***.
  - 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 x_i is to the classification decision,

    positive ➔ the feature is associated with the class

    negative ➔ the feature is not associated with the class
  - 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 inputs.

- **Test:** for given a test example x, we compute P(y|x) and return the higher probability label y=1 (member of class) or y=0 (not member of class).

Logistic Regression

weighted sum of the evidence for the class

- To make a 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.
- The resulting number z expresses the **weighted sum of the evidence for the class**.

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

- In short, we can write the weighted sum of the evidence for the class as follows:

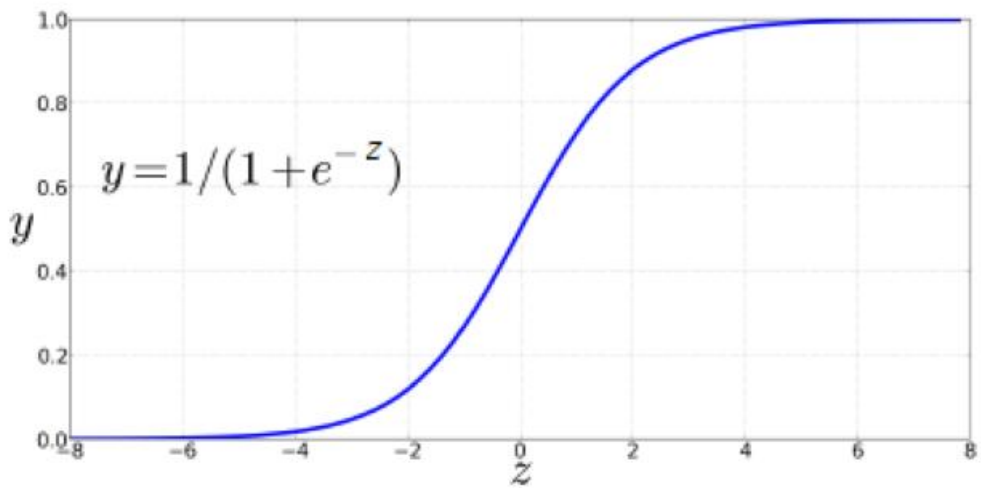
$$z = w \cdot x + b$$

- The weighted sum of the evidence z is *not a probability value*, it ranges from -∞ to +∞.

Logistic Regression

sigmoid function

- To create a probability, we'll pass z through the **sigmoid function**, ***σ*(z).**
- The **sigmoid function** is also called the **logistic function**.
- The sigmoid function $y = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range [0,1].



$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$

$\sigma(4)=1/(1+e^{-4})=0.982$
$\sigma(3)=1/(1+e^{-3})=0.953$
$\sigma(2)=1/(1+e^{-2})=0.881$
$\sigma(1)=1/(1+e^{-1})=0.731$
$\sigma(0)=1/(1+e^{-0})=0.5$
$\sigma(-1)=1/(1+e^{1})=0.269$
$\sigma(-2)=1/(1+e^{2})=0.119$
$\sigma(-3)=1/(1+e^{3})=0.047$
$\sigma(-4)=1/(1+e^{4})=0.018$

Logistic Regression

sigmoid function

- The sigmoid function has a number of advantages:
  - It maps a real-valued number into range [0,1], which is just what we want for a probability.
  - It tends to squash outlier values toward 0 or 1.
  - And it's differentiable, This will be handy for learning.
- If we apply the sigmoid to the sum of the weighted features, we get a number between 0 and 1.
- To make it a probability, we just need to make sure that the two cases, P(y=1) and P(y=0), sum to 1.

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

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

Logistic Regression

sigmoid function vs logit function

- The sigmoid function has the property: $\mathbf{1 - \sigma(x) = \sigma(-x)}$

$$\mathbf{1 - \sigma(x) = \frac{e^{-x}}{1 + e^{-x}} = \frac{e^{-x}}{1 + \frac{1}{e^{x}}} = \frac{e^{x} * \quad e^{-x}}{1 + e^{x}} = \frac{1}{1 + e^{x}} = \sigma(-x)}$$

- The input **z=wx+b** to the sigmoid function is often called **logit**.
- **Logit** function is the inverse of the sigmoid function.
- **Logit** function is the log of the odds ratio $\frac{p}{1-p}$

$$\mathrm{logit(p)} = \sigma^{-1}(\mathrm{p}) = \ln \frac{\mathrm{p}}{1 - \mathrm{p}}$$

$$\sigma(\mathrm{wx + b}) = \frac{1}{1 + \mathrm{e}^{-(\mathrm{wx+b})}} \qquad \mathrm{logit}\big(\sigma(\mathrm{wx + b})\big) = \mathrm{wx + b}$$

$$\mathrm{logit}(\sigma(z)) = ln \frac{\sigma(z)}{1-\sigma(z)} = ln \frac{1/(1+e^{-z})}{e^{-z}/(1+e^{-z})} = ln \frac{1}{e^{-z}} = ln\, e^{z} = z$$

Logistic Regression

decision boundary for classification

- We have an algorithm that given an instance x computes the probability P(y=1|x).

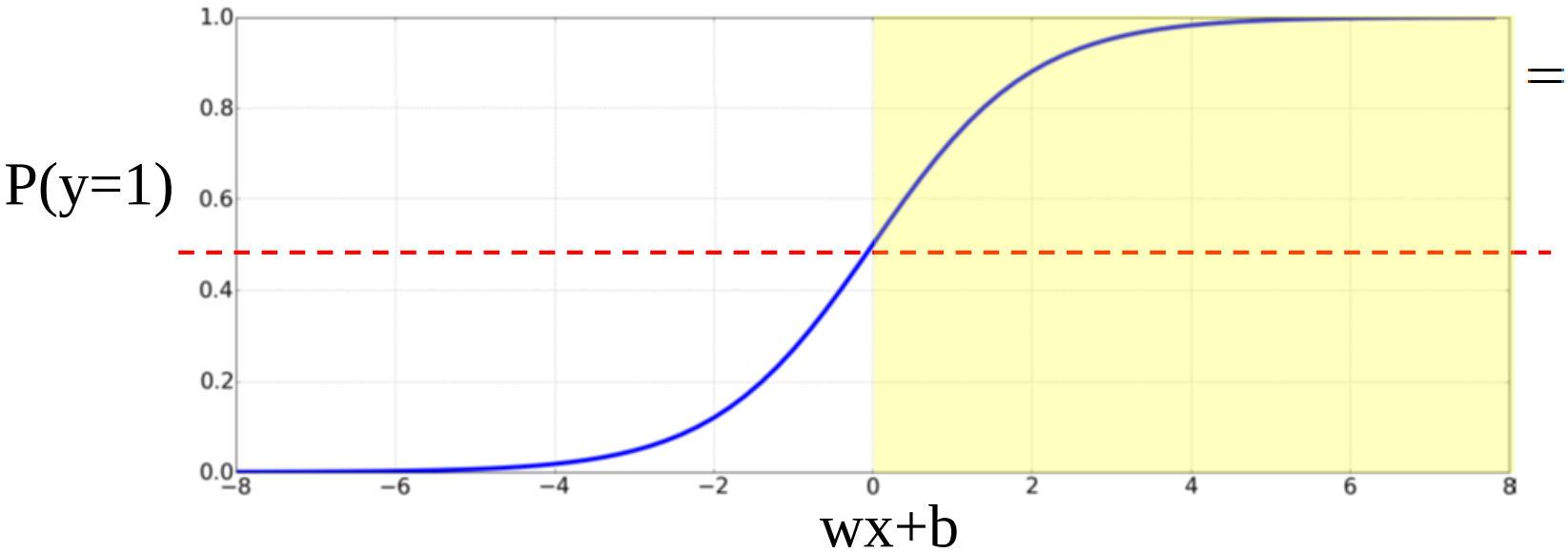
- When P(y=1|x) is more than 0.5 (**decision boundary**), the **estimated class** will be 1 (indicating that the instance x belongs to the class).

- Turning a probability into a classifier

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

Logistic Regression
decision boundary for classification

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



estimated class $\hat{\mathbf{y}} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$ $\quad$ $\begin{matrix} \textbf{if } \mathbf{w.x+b} > \mathbf{0} \\ \textbf{if } \mathbf{w.x+b} \leq \mathbf{0} \end{matrix}$

Logistic Regression :

Example on Sentiment Classification

Logistic Regression

Example: sentiment classification

- Suppose we are doing binary sentiment classification on movie review text, and we want to assign the sentiment class + or - to a review document *doc*.

- Each input observation (document) will be represented by 6 features of the input:

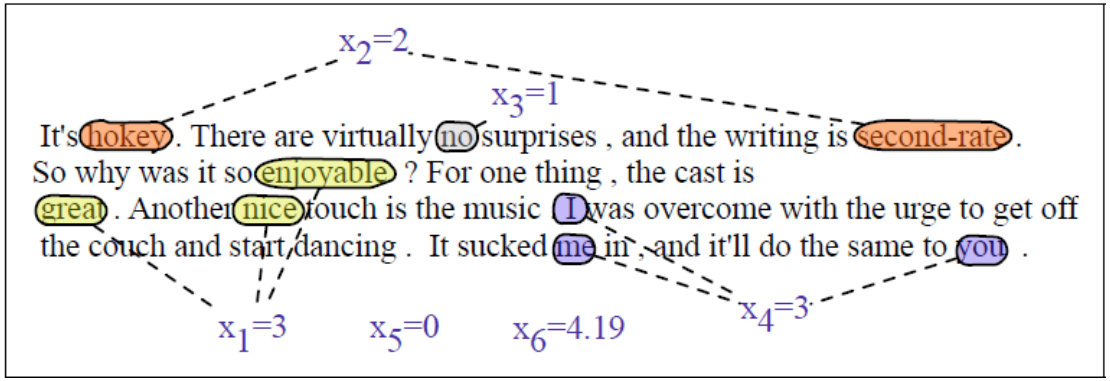
Var	Definition
x_1	count(positive lexicon words $\in$ doc)
x_2	count(negative lexicon words $\in$ doc)
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise}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

Logistic Regression

Example: sentiment classification

- Let's assume for the moment that we've already learned a real-valued weight for each of 6 features.
  - The 6 weights corresponding to the 6 features are [2.5, -5.0, -1.2, 0.5, 2.0, 0.7], and the bias term b = 0.1.
  - For example, the weight w_1 indicates how the number of positive lexicon words (great, nice, enjoyable, etc.) is important to a positive sentiment decision.

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



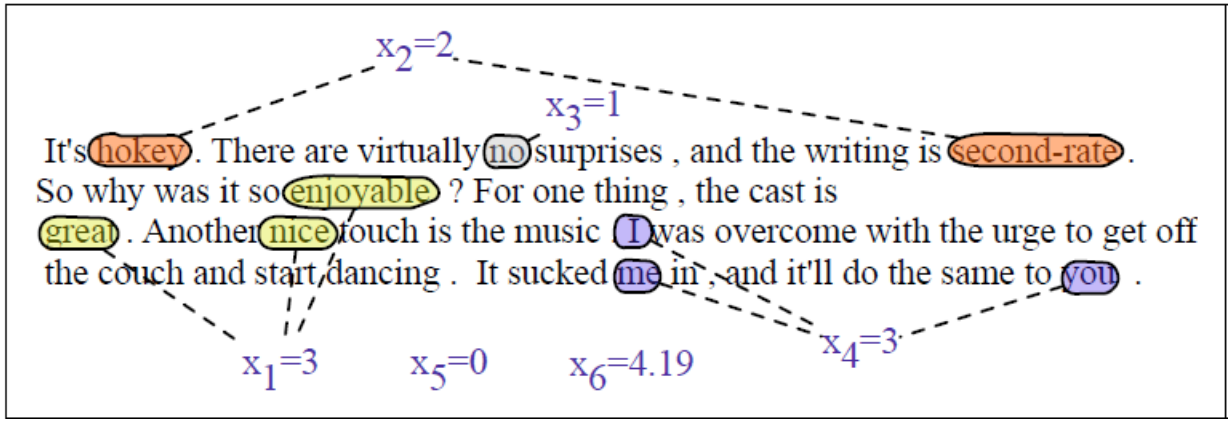
Var	Definition	Value
x_1	count(positive lexicon words $\in$ doc)	3
x_2	count(negative lexicon words $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc}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

Logistic Regression

Example: sentiment classification

- Given these 6 features and the input review x, P(+|x) and P(-|x) can be computed:

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



Logistic Regression: Learning

Logistic Regression: Learning

- How are the weights w and bias b (the parameters of the model) learned?
  - Logistic regression is an instance of supervised classification in which we know the correct label y (either 0 or 1) for each observation x.
  - The estimated value $\hat{y}$ is the system's estimate of the true y.
  - We want to learn parameters (w and b) that make $\hat{y}$ for each training observation as close as possible to the true y.

- In order to learn the parameters of the model, we require two things:

1. **A metric for how close the current label ($\hat{y}$) is to the true gold label y**.
   - The distance between the system output and the gold truth is called as the **loss function** or **the cost function**.
   - The loss function that is commonly used for logistic regression is the **cross-entropy loss**.
2. **An optimization algorithm for iteratively updating the weights so as to minimize this loss function.**
   - The standard algorithm for this is **gradient descent** (or **stochastic gradient descent**)

Logistic Regression: Learning

cross-entropy loss function

- We need a loss function that expresses, for an observation x, how close the classifier output ($\hat{\mathbf{y}} = \boldsymbol{\sigma}(\mathbf{w}.\mathbf{x} + \mathbf{b})$) is to the correct output (y, which is 0 or 1):

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

- For logistic regression, we will use a loss function known as **conditional maximum likelihood estimation** that prefers the correct class labels of the training example 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.
  - The resulting loss function is the negative log likelihood loss, generally called the **cross entropy loss.**
    - We do not prefer to use the *mean squared error* as a loss function because it is harder to optimize for probabilistic classification (it is not convex).

Logistic Regression: Learning

cross-entropy loss function

- We'd like to learn weights that MAXIMIZE the probability of the correct label p(y|x).
- Since there are only two discrete outcomes (1 or 0), we can express the probability p(y|x) as follows:

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

p(y|x)= $\hat{y}$ when y=1 and
p(y|x)=1-$\hat{y}$ when y=0

- Take the log of both sides (whatever values maximize a probability will also maximize the log of the probability):

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

  – This describes a log likelihood that should be maximized.

Logistic Regression: Learning

cross-entropy loss function

- To obtain **cross-entropy loss function for one example**, we change the sign in the equation and we have to MINIMIZE this L_{CE} function:

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

- Plug in the definition of $\hat{y} = \sigma(w.x + b)$:

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

  - A perfect classifier would assign probability 1 to the correct outcome (y=1 or y=0) and probability 0 to the incorrect outcome.
  - Higher $\hat{y}$ (close to 1) ➔ better classifier; Lower $\hat{y}$ is (close to 0) ➔ worse classifier.
    - The negative log of this probability is a convenient loss metric since it goes from 0 to infinity.
    - This loss function also insures that as probability of the correct answer is maximized, the probability of the incorrect answer is minimized;

Logistic Regression: Learning

cross-entropy loss function

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

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

y=1 ➔ $L_{CE}(\hat{y},1) = -\log(\hat{y}) = -\log(\sigma(w.x+b))$

y=0 ➔ $L_{CE}(\hat{y},0) = -\log(1-\hat{y}) = -\log(1-\sigma(w.x+b))$

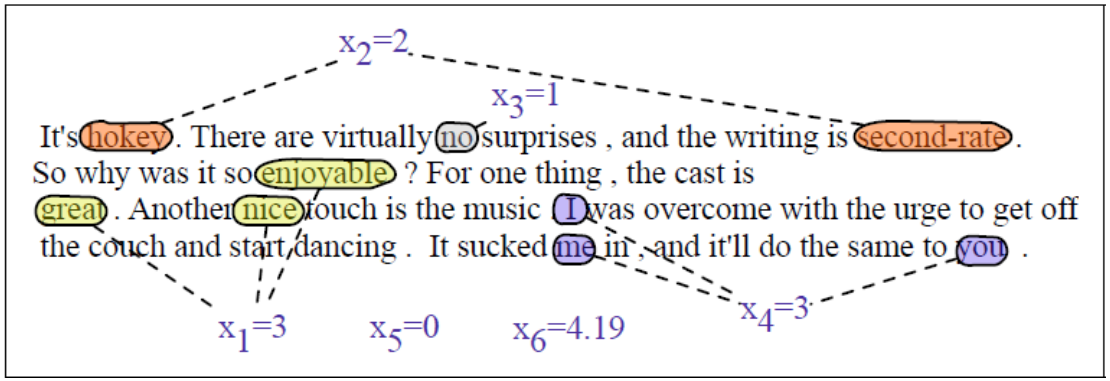
$L_{CE}(1.0,1) = 0$	$L_{CE}(0.0,0) = 0$
$L_{CE}(0.8,1) = 0.223$	$L_{CE}(0.2,0) = 0.223$
$L_{CE}(0.5,1) = 0.693$	$L_{CE}(0.5,0) = 0.693$
$L_{CE}(0.2,1) = 1.609$	$L_{CE}(0.8,0) = 1.609$
$L_{CE}(0.1,1) = 2.303$	$L_{CE}(0.9,0) = 2.303$
$L_{CE}(0.01,1) = 4.605$	$L_{CE}(0.99,0) = 4.605$

Logistic Regression: Learning

cross-entropy loss function: example

- We want **loss** to be
  - **smaller** if the model estimate is close to correct,
  - **bigger** if model is confused

- Sentiment example:



Current weights of the model:

weights: [2.5,-5.0,-1.2,0.5,2.0,0.7], and the bias term b = 0.1.

- What will be the loss value for this example if its true value is y=1 or y=0?

Logistic Regression: Learning

cross-entropy loss function: example

- If its true value is y=1 or y=0, the model produces following probabilty values with the current weights.

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

- If its true value is y=1, the loss will be:

$$
\begin{aligned}
L_{\mathrm{CE}}(\hat{y},y) &= -[y\log\sigma(\mathbf{w}\cdot\mathbf{x}+b)+(1-y)\log(1-\sigma(\mathbf{w}\cdot\mathbf{x}+b))] \\
&= -[\log\sigma(\mathbf{w}\cdot\mathbf{x}+b)] \;=\; -\log(.70) \;=\; .36
\end{aligned}
$$

- If its true value y=0:

$$
\begin{aligned}
L_{\mathrm{CE}}(\hat{y},y) &= -[y\log\sigma(\mathbf{w}\cdot\mathbf{x}+b)+(1-y)\log(1-\sigma(\mathbf{w}\cdot\mathbf{x}+b))] \\
&= -[\log(1-\sigma(\mathbf{w}\cdot\mathbf{x}+b))] \;=-\log(.30) \;=\; 1.2
\end{aligned}
$$

Logistic Regression: Learning

Gradient Descent

- Our goal with the **gradient descent** is to find the **optimal weights** which minimize the **loss function**.
- The loss function L is *parameterized by the weights*, which we'll refer to as θ (in the logistic regression θ = (w,b), and the **optimal weights** are:

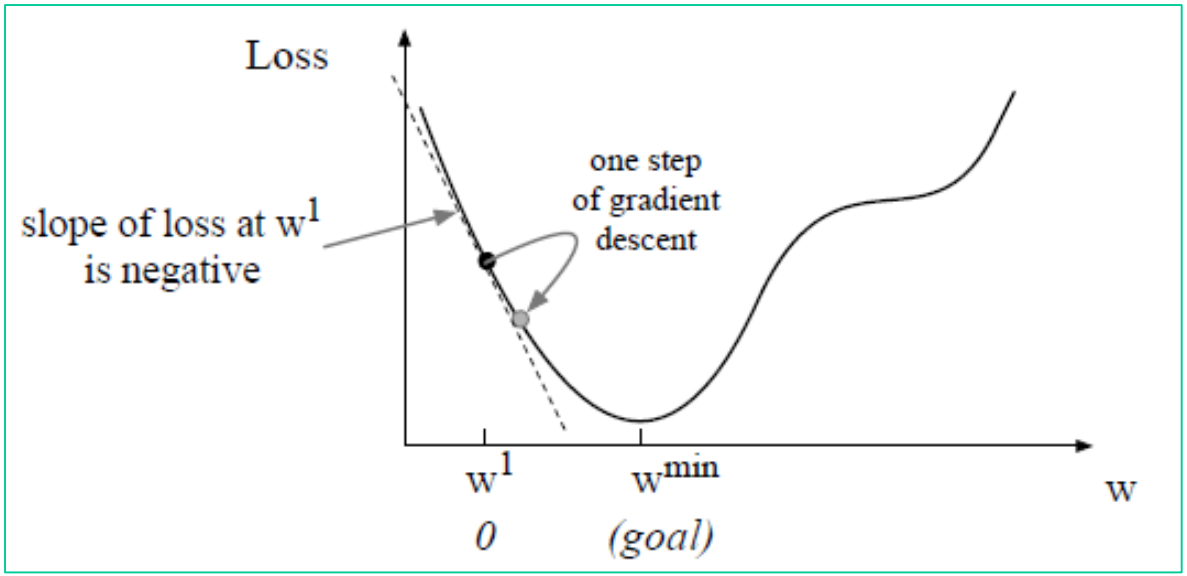
$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\mathrm{CE}}(f(x^{(i)};\theta), y^{(i)})$$

- **Gradient descent** is a method that finds a minimum of a function by figuring out in which direction (in the space of the parameters θ) the function's slope is rising the most steeply, and moving in the opposite direction.
- For logistic regression, the ***loss function is conveniently convex***.
  - A convex function has just one minimum; there are no local minima to get stuck in, so **gradient descent** starting from any point is guaranteed to find the minimum.

Logistic Regression: Learning

Gradient Descent

- Although the **gradient descent algorithm** is designed for *direction vectors*, let's first consider the algorithm for a *single value w*.
- After the random initial value w^1 (normally 0), the **gradient descent algorithm** tell us our direction at the next iteration to reach the minimum.
  - move w in positive direction (to right) when slope of loss is negative (making w^2 bigger than w^1) or
  - move w in negative direction (to left) when slope of loss is positive (making w^2 smaller than w^1).



- The **gradient** of a function of many variables is a vector pointing in the direction of the greatest increase in a function.
- **Gradient Descent:** Find the gradient of the loss function at the current point and move in the opposite direction.

Logistic Regression: Learning

Gradient Descent

- In order to reach to the minimum, the **gradient descent algorithm finds the gradient of the loss function** at the current point and it **moves in the opposite direction**.

- The **gradient** is the increase direction, and it is equal to the **slope:** $\frac{\mathbf{d}}{\mathbf{dw}}\mathbf{L}(\mathbf{f}(\mathbf{x};\mathbf{w}),\mathbf{y})$

- The **magnitude of the amount to move in gradient descent** is the value of the slope $\frac{\mathbf{d}}{\mathbf{dw}}\mathbf{L}(\mathbf{f}(\mathbf{x};\mathbf{w}),\mathbf{y})$ weighted by a **learning rate** $\boldsymbol{\eta}$.
  - Learning rate $\boldsymbol{\eta}$ is a hyperparameter.

- Update weight w by subtracting (the learning rate times the gradient)

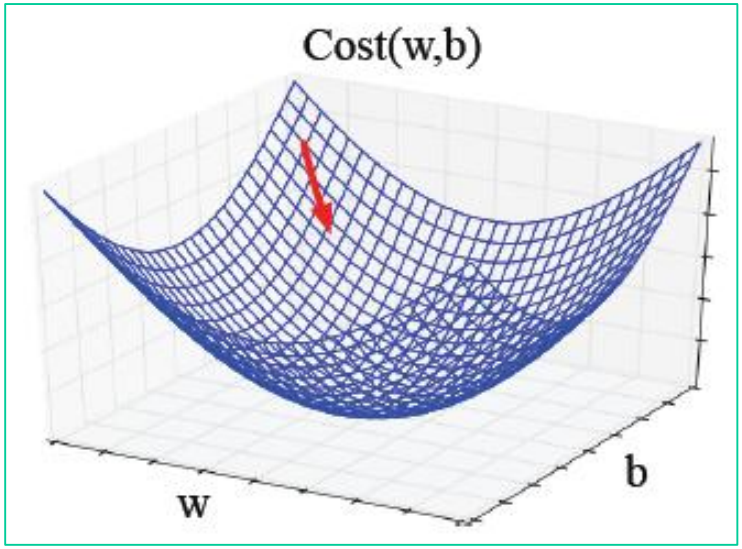
$$w^{t+1} = w^t - \boxed{\eta}\boxed{\frac{d}{dw}L(f(x;w),y)}$$

learning rate **gradient** of the loss function

Logistic Regression: Learning

Gradient Descent – N Dimensions

- When we have a weight vector, we want to know where in the N-dimensional space (N parameters in θ) we should move.
- The **gradient** is just such a vector which expresses the *directional components of the sharpest slope along each of those N dimensions*.
- Visualization of the gradient vector in two dimensions w and b.



Logistic Regression: Learning

Gradient Descent – N Dimensions

- For each variable w_i in w, the **gradient** will have ***a component that tells us the slope with respect to that variable***.
  - "How much would a small change in that variable w_i influence the total loss function L?"
- In each dimension w_i, we express the slope as a **partial derivative** $\frac{\partial}{\partial w_i}$ **of the loss function**.
- The **gradient** is then defined as **a vector of these partials**.

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

where f(x;θ) is $\hat{y}$

Logistic Regression: Learning

Gradient Descent - N Dimensions

- The equation for updating **θ** based on the **gradient** is:

$$\theta^{t+1} = \theta^{t} - \eta \nabla L(f(x;\theta),y)$$

- In order to update **θ**, we need a definition for the gradient $\nabla L(f(x;\theta),y)$.
- The cross-entropy loss function for logistic regression is:

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

- The partial derivative of this function for one observation vector x is:

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

Logistic Regression: Learning

Gradient Descent – Math. for Partial Derivation of L_{CE}

$$\mathbf{L_{CE}(\hat{y},y) = -\ (\ y.\ln(\sigma(z)) + (1-y).\ln(1-\sigma(z))\ )}$$

where $z=w.x+b$ $\hat{y}=\sigma(z))$

$$\frac{\partial L_{CE}(\hat{y},y)}{\partial w_j} = \frac{\partial L_{CE}(\hat{y},y)}{\partial \sigma(z)}\ \frac{\partial \sigma(z)}{\partial z}\ \frac{\partial z}{\partial w_j}$$

$$\frac{\partial L_{CE}(\hat{y},y)}{\partial \sigma(z)} = -(y.\frac{1}{\sigma(z)} + (1-y).\frac{-1}{1-\sigma(z)}) \qquad \frac{\partial \sigma(z)}{\partial z} = \sigma(z).(1-\sigma(z)) \qquad \frac{\partial z}{\partial w_j} = x_j$$

$$\frac{\partial L_{CE}(\hat{y},y)}{\partial w_j} = -\left(y.\frac{1}{\sigma(z)} + (1-y).\frac{-1}{1-\sigma(z)}\right).\sigma(z).(1-\sigma(z)).x_j$$

$$\frac{\partial L_{CE}(\hat{y},y)}{\partial w_j} = -\Big(y.(1-\sigma(z)) + (1-y).\big(-\sigma(z)\big)\Big).x_j - \big(y-\sigma(z)\big).x_j = (\sigma(z)-y).x_j = (\hat{y}-y).x_j$$

$$\mathbf{\frac{\partial L_{CE}(\hat{y},y)}{\partial w_j} = (\hat{y}-y).x_j = (\sigma(w.x+b)-y).x_j}$$

Logistic Regression: Learning

Gradient Descent - N Dimensions

- The **cost function** (or **loss function**) for all m examples in the training set.

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

- **Gradient** for the whole training set is:

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

Logistic Regression: Learning

Stochastic Gradient Descent Algorithm

- **Stochastic gradient descent** minimizes the loss function by computing its gradient after each training example, and nudging **θ** in the right direction.

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),..., x^(m)
    #        y is the set of training outputs (labels) y^(1), y^(2),..., y^(m)

θ ← 0
repeat til done   // T times or some stopping condition is reached
  For each training tuple (x^(i), y^(i)) (in random order)
     1. Optional (for reporting):          # How are we doing on this tuple?
        Compute ŷ^(i) = f(x^(i); θ)        # What is our estimated output ŷ?
        Compute the loss L(ŷ^(i), y^(i))   # How far off is ŷ^(i) from the true output y^(i)?
     2. g ← ∇_θ L(f(x^(i); θ), y^(i))      # How should we move θ to maximize loss?
     3. θ ← θ − η g                        # Go the other way instead
return θ
```

Logistic Regression: Learning

Stochastic Gradient Descent: Example – sentiment classification

- We will give some steps of stochastic gradient descent algorithm for a simplified version of sentiment classification task.
  - It sees a single observation x, whose correct value is y = 1 (this is a positive review), and
  - There are only two features:
    - $x_1 = 3$ (count of positive lexicon words)
    - $x_2 = 2$ (count of negative lexicon words)

- Initial weights and bias in $\boldsymbol{\theta}^1$, and the learning rate $\boldsymbol{\eta}$ are:

$$w_1 = w_2 = b = 0$$
$$\eta = 0.1$$

Logistic Regression: Learning

Stochastic Gradient Descent: Example – sentiment classification

- The **single update step:**

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$$

- Since there are three parameters, so the gradient vector has 3 dimensions, for w_1, w_2, and b. We can compute the gradient as follows:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(w,b)}{\partial w_1} \\ \frac{\partial L_{CE}(w,b)}{\partial w_2} \\ \frac{\partial L_{CE}(w,b)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

- we compute $\boldsymbol{\theta}^2$ by moving $\boldsymbol{\theta}^1$ in the opposite direction from the gradient:

$$\theta^2 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

$$\mathbf{Q}^{t+1} = \overset{\mathbf{Q}^t}{\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix}} - \eta \overset{\textbf{gradient}}{\begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix}}$$

Logistic Regression: Learning

Stochastic Gradient Descent: Example – sentiment classification

$$Q^{t+1} = \overset{Q^t}{\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix}} - \eta \overset{\text{gradient}}{\begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix}}$$

weights	examples	**Initial Weights: Θ1**	Ex1	gradient	**Weights: Θ2**
w1	x1	**0.000**	3.000	-1.500	**0.150**
w2	x2	**0.000**	2.000	-1.000	**0.100**
b	x0 (=1)	**0.000**	1.000	-0.500	**0.050**
	y: class		1.000		
w.x+b			0.000		
sig(w.x+b)			0.500		
eta	**0.1**				

Logistic Regression: Learning

Stochastic Gradient Descent: Example – sentiment classification

$$Q^{t+1} = \overset{Q^t}{\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix}} - \eta \overset{\text{gradient}}{\begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix}}$$

weights	examples	**Weights: Θ2**	Ex2	gradient	**Weights: Θ3**	Ex3	gradient	**Weights: Θ4**
w1	x1	**0.150**	4.000	-1.372	**0.287**	0.000	0.000	**0.287**
w2	x2	**0.100**	0.000	0.000	**0.100**	4.000	2.475	**-0.148**
b	x0 (=1)	**0.050**	1.000	-0.343	**0.084**	1.000	0.619	**0.022**
	y: class		1.000			0.000		
w.x+b			0.650			0.484		
sig(w.x+b)			0.657			0.619		
eta	**0.1**							

Logistic Regression: Learning

Stochastic Gradient Descent: Example – sentiment classification

$$Q^{t+1} = \overset{Q^t}{\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix}} - \eta \overset{\text{gradient}}{\begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix}}$$

weights	examples	**Weights: Θ4**	Ex4	gradient	**Weights: Θ5**	Ex5	gradient	**Weights: Θ6**
w1	x1	**0.287**	3.000	-0.971	**0.384**	0.000	0.000	**0.384**
w2	x2	**-0.148**	1.000	-0.324	**-0.115**	3.000	1.284	**-0.243**
b	x0 (=1)	**0.022**	1.000	-0.324	**0.055**	1.000	0.428	**0.012**
	y: class		1.000			0.000		
w.x+b			0.737			-0.291		
sig(w.x+b)			0.676			0.428		
eta	**0.1**							

Logistic Regression: Learning

Mini-Batch Training

- **Stochastic gradient descent** chooses a single random example at a time.
  - That can result in choppy movements
- We compute the gradient over the entire dataset.
  - **batch gradient descent**
  - By seeing so many examples, batch training over the entire dataset offers a superb estimate of which direction to move the weights, at the cost of spending a lot of time processing every single example in the training set to compute this perfect direction.
- More common to compute gradient over mini-batches of m examples of training instances.
  - **mini-batch training**: m examples (512, or 1024) is less than size of the dataset.
  - m=1 ➔ **stochastic gradient descent**
  - m is the size of the dataset ➔ **gradient descent**

Logistic Regression: Learning

Mini-Batch Training

- **Cost function for the mini-batch** of m examples is the average loss for each example:

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

- The **mini-batch gradient** is the average of the individual gradients:

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

Logistic Regression: Learning

Overfitting and Regularization

- The ***weights for features will attempt to perfectly fit details of the training set, modeling noisy factors that just accidentally correlate with the class***.
- This problem is called **overfitting**.
  - A good model should be able to **generalize** well from the training data to the unseen test set, but a model that **overfits** will have poor generalization.

- One way to **avoid overfitting**, adding a **regularization term** to the objective function:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) - \alpha R(\theta)$$

  - The new component, R(θ) is called a regularization term, and is used to penalize large weights.

Multinomial Logistic Regression

- In some classification tasks, we need more than two classes.
  - Positive/negative/neutral in sentiment analysis
  - Parts of speech (noun, verb, adjective, adverb, preposition, etc.)
  - Classify emergency SMSs into different actionable classes

- In such cases, we can use **multinominal logistic regression**, also it is also called as **softmax regression** (or it is also called as **maxent classifier**).
  - **logistic regression** will just mean binary classification (2 output classes)

- In **multinominal logistic regression,** the target y is a variable that ranges over more than two classes.
  - We want label an observation with a class $c \in C = \{c_1, c_2, \ldots, c_k\}$ where $k>2$, and only one of these classes is correct (*hard classification – observation cannot be in multiple classes)*
  - We want to know the probability of y being in each potential class $c \in C$, $p(y=c|x)$.

Multinomial Logistic Regression

Softmax

- The probability of everything must still sum to 1

  $$P(\text{positive}|\text{doc}) + P(\text{negative}|\text{doc}) + P(\text{neutral}|\text{doc}) = 1$$

- The **multinominal logistic classifier** uses a generalization of the sigmoid function, called as the **softmax function**, to compute the probability $p(y=c|x)$.
  - The **softmax function** takes a vector $z = [z_1, z_2, \ldots, z_k]$ of k arbitrary values and maps them to a probability distribution,
  - each value in the range (0,1]
  - all the values summing to 1.

- Like the sigmoid, the softmax function is also an exponential function.

Multinomial Logistic Regression

Softmax

- The **softmax function** turns a vector z = [z_1, z_2, ... , z_k] of k arbitrary values into probabilities.

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \le i \le k$$

where $\exp(z_i) = e^{z_i}$

- The denominator $\sum_{j=1}^{k} \exp(z_j)$ is used to normalize all the values into probabilities.

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

Multinomial Logistic Regression

Softmax

- For a given vector

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

- The the resulting softmax values is

softmax(z) = [0.055, 0.090, 0.007, 0.100, 0.738, 0.010]

using the softmax function

z	exp(z)	softmax(z)
0.6	1.822	0.055
1.1	3.004	0.090
-1.5	0.223	0.007
1.2	3.320	0.100
3.2	24.533	0.738
-1.1	0.333	0.010
sum	33.235	

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

Multinomial Logistic Regression

Softmax in moltinominal logistic regression

- Input for the **softmax function** is still the dot product between weight vector w (including bias b) and input vector x.

- But now we will need separate weight vectors for each of the k classes.

- The probability of each of our output classes y=c can thus be computed as:

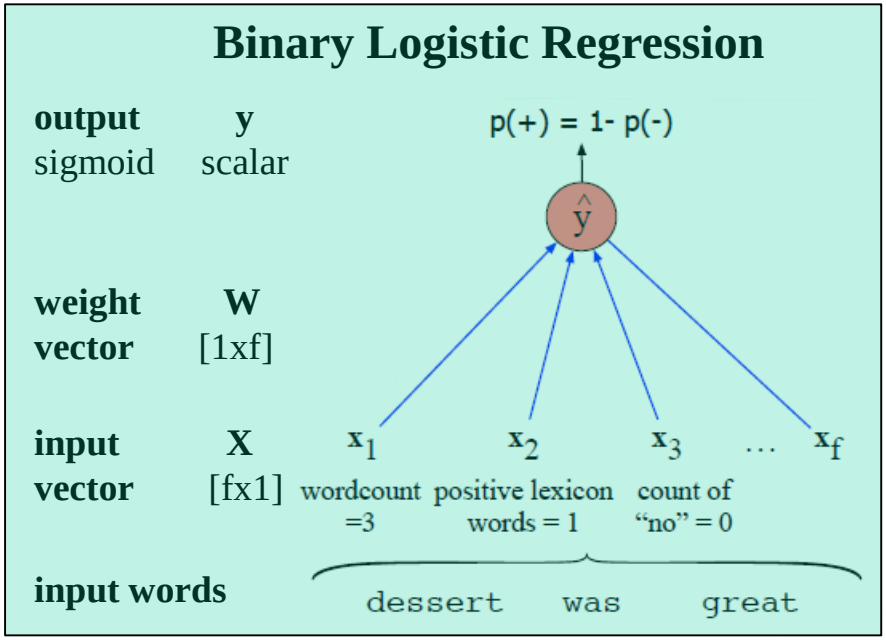
$$p(y=c|x) = \frac{\exp(w_c \cdot x + b_c)}{\sum_{j=1}^{k} \exp(w_j \cdot x + b_j)}$$

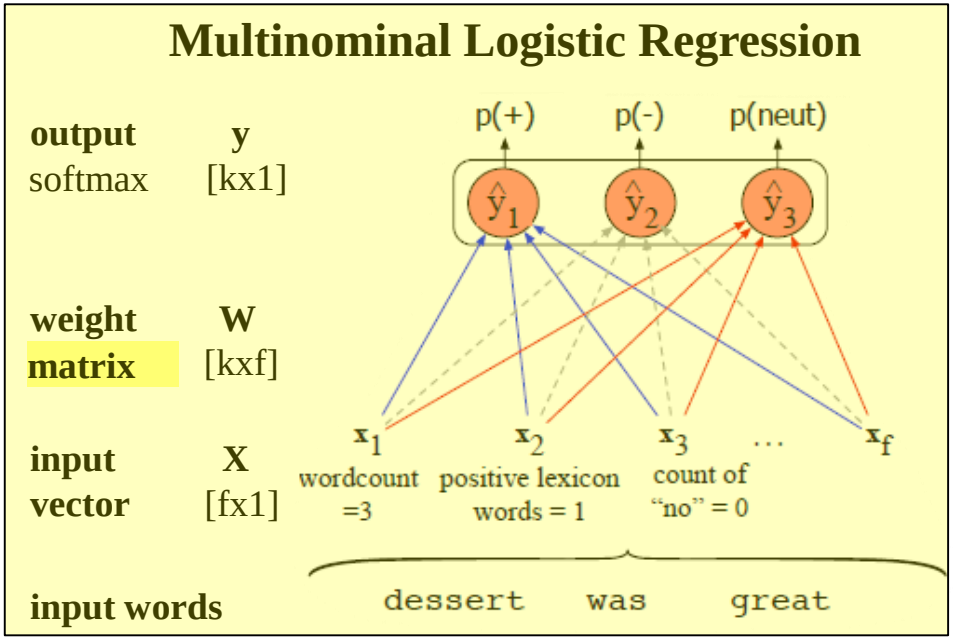
Multinomial Logistic Regression

Features in binary versus moltinominal logistic regression

- **Binary logistic regression** uses a single weight vector w, and has a scalar output $\hat{\mathbf{y}}$.

- In **multinomial logistic regression** we have k separate weight vectors corresponding to the k classes, all packed into a single weight matrix W, and a vector output $\hat{y}=[\hat{y}_1,\ldots,\hat{y}_k]$.





Multinomial Logistic Regression

Features in binary versus moltinominal logistic regression

- Consider the following feature for the sentiment classification:

$$x_5 = \begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$$

- In binary logistic regression, there is a single weight w_5 for this feature.
  - A positive weight w_5 on a feature influences the classifier toward y = 1 (positive sentiment) and a negative weight influences it toward y = 0 (negative sentiment) with the absolute value indicating how important the feature is.
- In multinomial logistic regression, there are separate weights for each class, a feature can be evidence for or against each individual class.
- In 3-way multiclass sentiment classification, for example, we must assign each document one of the 3 classes +, -, or 0 (neutral).
  - Now this feature will have a negative weight for 0 documents, and a positive weight for + or – documents according to follwing weightd:

Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	−5.3

Multinomial Logistic Regression

Learning

- **Cross-entropy loss** for **binary logistic regression**

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

- For **multinomial logistic regression** we'll represent both **y** and $\hat{\mathbf{y}}$ as vectors.
  - The true label y is a vector with K elements, each corresponding to a class, with $y_c = 1$ if the correct class is c, with all other elements of y being 0.
  - Our classifier will produce an estimate vector with K elements $\hat{\mathbf{y}}$, each element $\hat{\mathbf{y}}_{\mathbf{k}}$ represents the estimated probability $p(y_k=1|x)$.

- **Cross-entropy loss** for **multinomial logistic regression**

$$L_{\mathrm{CE}}(\hat{\mathbf{y}},\mathbf{y}) \;=\; -\sum_{k=1}^{K} y_k \log\hat{y}_k$$

(where c is the correct class)

$$= -\log\hat{y}_c \;=\; -\log\hat{p}(y_c=1|\mathbf{x}) \;=\; -\log\frac{\exp(\mathbf{w_c}\cdot\mathbf{x}+b_c)}{\sum_{j=1}^{K}\exp(\mathbf{w_j}\cdot\mathbf{x}+b_j)}$$

Multinomial Logistic Regression

Learning

- **Gradient** (for a single example) in **multinomial logistic regression**
  - This derivative turns out to be just the difference between the true value for the class k (which is either 1 or 0) and the probability the classifier outputs for class k, weighted by the value of the input x_i corresponding to the ith element of the weight vector for class k:

$$
\begin{aligned}
\frac{\partial L_{\mathrm{CE}}}{\partial w_{k,i}} &= -(y_k - \hat{y}_k)x_i \\
&= -(y_k - p(y_k = 1|x))x_i \\
&= -\left(y_k - \frac{\exp(\mathbf{w_k} \cdot \mathbf{x} + b_k)}{\sum_{j=1}^{K} \exp(\mathbf{w_j} \cdot \mathbf{x} + b_j)}\right) x_i
\end{aligned}
$$

$$
\begin{aligned}
L_{\mathrm{CE}}(\hat{\mathbf{y}}, \mathbf{y}) &= -\sum_{k=1}^{K} y_k \log \hat{y}_k \\
&= -\log \hat{y}_c = -\log \hat{p}(y_c = 1|\mathbf{x}) = -\log \frac{\exp(\mathbf{w_c} \cdot \mathbf{x} + b_c)}{\sum_{j=1}^{K} \exp(\mathbf{w_j} \cdot \mathbf{x} + b_j)}
\end{aligned}
$$

(where c is the correct class)

Multinomial Logistic Regression: Learning

Gradient Descent – Math. for Partial Derivation of L_{CE}

$$L_{CE}(\hat{y},y) = -\sum_{k=1}^{K} y_k \log\hat{y}_k = -\log\hat{y}_c = -\log(\hat{p}(y_c = 1|x)) = -\log\frac{\exp(z_c)}{\sum_{k=1}^{K}\exp(z_k)}$$

where c is correct class
$z_k = w_k.x + b_k$

$$L_{CE}(\hat{y},y) = -\log\frac{\exp(z_c)}{\sum_{k=1}^{K}\exp(z_k)} = -\log(\exp(z_c)) + \log\left(\sum_{k=1}^{K}\exp(z_k)\right) = -z_c + \log\left(\sum_{k=1}^{K}\exp(z_k)\right)$$

$$= -(w_c.x + bc) + \log\left(\sum_{k=1}^{K}\exp(w_k.x + b_k)\right)$$

when k≠c: $\frac{\partial L_{CE}(\hat{y},y)}{\partial w_{k,i}} = \frac{\exp(w_k.x+b_k).x_i}{\sum_{j=1}^{K}\exp(w_j.x+bj)}$

when k=c: $\frac{\partial L_{CE}(\hat{y},y)}{\partial w_{k,i}} = -x_i + \frac{\exp(w_k.x+b_k).x_i}{\sum_{j=1}^{K}\exp(w_j.x+b_j)}$

Since y_k is 1 when k=c, and y_k is 0 when k ≠ c

$$\boxed{\frac{\partial L_{CE}(\hat{y},y)}{\partial w_{k,i}} = -\left(y_k - \frac{\exp(w_k.x+b_k)}{\sum_{j=1}^{K}\exp(w_j.x+b_j)}\right).x_i = -(y_k - \hat{y}_k).x_i}$$

$$\frac{\partial L_{CE}(\hat{y},y)}{\partial w_{k,i}} = \frac{\partial \log\left(\sum_{j=1}^{K}\exp(w_j.x+b_j)\right)}{\partial w_{k,i}}$$

$$= \frac{1}{\sum_{j=1}^{K}\exp(w_j.x+b_j)}.\frac{\partial \sum_{j=1}^{K}\exp(w_j.x+b_j)}{\partial w_{k,i}}$$

$$= \frac{1}{\sum_{j=1}^{K}\exp(w_j.x+b_j)}.\exp(w_k.x+b_k)\frac{\partial(w_k.x+b_k)}{\partial w_{k,i}}$$

$$= \frac{\exp(w_k.x+b_k)}{\sum_{j=1}^{K}\exp(w_j.x+b_j)}.x_i$$

$$\frac{\partial \log(f(x))}{\partial x} = \frac{1}{f(x)}.\frac{\partial f(x)}{\partial x}$$

$$\frac{\partial \exp(f(x))}{\partial x} = \exp(f(x)).\frac{\partial f(x)}{\partial x}$$

Multinomial Logistic Regression

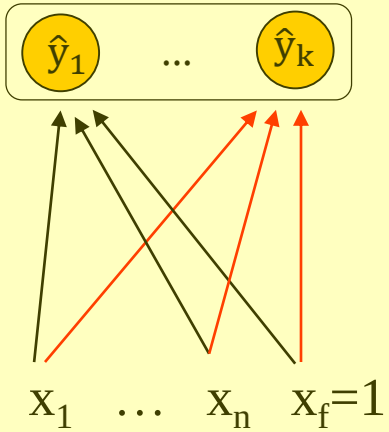
Learning: Example

Multinomial Logistic Regression

output $\hat{y}$
softmax [k×1]

weight **W**
matrix [k×f]

input **X**
vector [f×1]



x_f is the feature value for bias which is always 1.
f = n+1 i.e. n+1 features

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k}\exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k}\exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k}\exp(z_i)}\right]$$

estimated output $\hat{y}$:

$$\hat{y} = [\hat{y}_1, \ldots, \hat{y}_k] = \text{softmax}(W_{k\times f} \,.\, X_{f\times 1})$$

input vector: $X = [x_1, \ldots, x_n, x_f=1]$

weight matrix:

$$W = \begin{bmatrix} w_{1,1} & \cdots & w_{1,n} & b_1 \\ \vdots & & \vdots & \\ w_{k,1} & \cdots & w_{k,n} & b_n \end{bmatrix} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,f} \\ \vdots & & \vdots \\ w_{k,1} & \cdots & w_{k,f} \end{bmatrix}$$

$w_i = [w_{i,1} \quad \cdots \quad w_{i,f}]$ weight vector for class i.

correct output:

$y = [y_1, \ldots, y_k]$ only one y_i is 1 where i=c is the correct class; others are 0

Multinomial Logistic Regression

Learning: Example

Three features: $X = [x_1, x_2, x_3, x_f = 1]$ where f=4 and x_f (always 1) is the feature value for bias.

Three classes (+,-,neutral): $\hat{y} = [\hat{y}_1, \hat{y}_2, \hat{y}_3]$

Ex: X=(2,3,4) ➔ X=[2,3,4,1] i.e. $x_1=2$ $x_2=3$ $x_1=4$ $x_4=1$

positive example ➔ y=[1,0,0] i.e. $y_1=1$ $y_2=0$ $y_3=0$

X is a fx1 matrix $X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix}$

Learing rate η=0.1, Initial weight matrix $W^1 = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

$W^1 . X = [0,0,0]$

$\hat{y} = [\hat{y}_1, \hat{y}_2, \hat{y}_3] = \text{softmax}(W^1 . X) = \text{softmax}([0,0,0]) = [1/3, 1/3, 1/3]$ i.e. $\hat{y}_1 = 1/3$ $\hat{y}_2 = 1/3$ $\hat{y}_3 = 1/3$

$$\text{gradient} = \begin{bmatrix} \frac{\partial L_{ce}}{\partial w_{1,1}} & \frac{\partial L_{ce}}{\partial w_{1,2}} & \frac{\partial L_{ce}}{\partial w_{1,3}} & \frac{\partial L_{ce}}{\partial w_{1,4}} \\ \frac{\partial L_{ce}}{\partial w_{2,1}} & \frac{\partial L_{ce}}{\partial w_{2,2}} & \frac{\partial L_{ce}}{\partial w_{2,3}} & \frac{\partial L_{ce}}{\partial w_{2,4}} \\ \frac{\partial L_{ce}}{\partial w_{3,1}} & \frac{\partial L_{ce}}{\partial w_{3,2}} & \frac{\partial L_{ce}}{\partial 3} & \frac{\partial L_{ce}}{\partial w_{3,4}} \end{bmatrix} = \begin{bmatrix} -(y_1 - \hat{y}_1).x_1 & -(y_1 - \hat{y}_1).x_2 & -(y_1 - \hat{y}_1).x_3 & -(y_1 - \hat{y}_1).x_4 \\ -(y_2 - \hat{y}_2).x_1 & -(y_2 - \hat{y}_2).x_2 & -(y_2 - \hat{y}_2).x_3 & -(y_2 - \hat{y}_2).x_4 \\ -(y_3 - \hat{y}_3).x_1 & -(y_3 - \hat{y}_3).x_2 & -(y_3 - \hat{y}_3).x_3 & -(y_3 - \hat{y}_3).x_4 \end{bmatrix} = \begin{bmatrix} -\frac{2}{3}.2 & -\frac{2}{3}.3 & -\frac{2}{3}.4 & -\frac{2}{3}.1 \\ \frac{1}{3}.2 & \frac{1}{3}.3 & \frac{1}{3}.4 & \frac{1}{3}.1 \\ \frac{1}{3}.2 & \frac{1}{3}.3 & \frac{1}{3}.4 & \frac{1}{3}.1 \end{bmatrix}$$

$$W^2 = W^1 - \eta.\text{gradient} = \begin{bmatrix} \frac{0.4}{3} & \frac{0.6}{3} & \frac{0.8}{3} & \frac{0.2}{3} \\ -\frac{0.2}{3} & -\frac{0.3}{3} & -\frac{0.4}{3} & -\frac{0.1}{3} \\ -\frac{0.2}{3} & -\frac{0.3}{3} & -\frac{0.4}{3} & -\frac{0.1}{3} \end{bmatrix}$$

Logistic Regression: Summary

- **Logistic regression** is a supervised machine learning classifier that extracts real-valued features from the input, multiplies each by a weight, sums them, and passes the sum through a sigmoid function to generate a probability.
  - A threshold is used to make a decision.
- The weights (vector w and bias b) are learned from a labeled training set via a loss function, such as the **cross-entropy loss**, that must be minimized.
- Minimizing this loss function is a **convex optimization problem**, and iterative algorithms like **gradient descent** are used to find the ***optimal weights***.
- **Regularization** is used to avoid overfitting.
- **Logistic regression** can be used with two classes or with multiple classes (**multinomial logistic regression**).
- **Multinomial logistic regression** uses the **softmax function** to compute probabilities.