CSCI567 Machine Learning (Fall 2017)

Prof. Fei Sha

U of Southern California

Lecture on Sept. 7, 2017

Outline

1. Administration
2. Review of Last Lecture
3. Logistic regression
4. Summary

Outline

1. Administration
2. Review of Last Lecture
3. Logistic regression
4. Summary

Administrative stuff

- No more D-clearance will be granted.
- If you have not submitted Github handle or being enrolled into Piazza, please contact Michael Shindler at shindler@usc.edu *after* you complete the syllabus quiz:
  `https://goo.gl/forms/hkbPoLOsoVasBAxU2`
- HW1 to be released tomorrow morning
- After Friday, we will clean up Github and Piazza for those who are not enrolled.

Outline

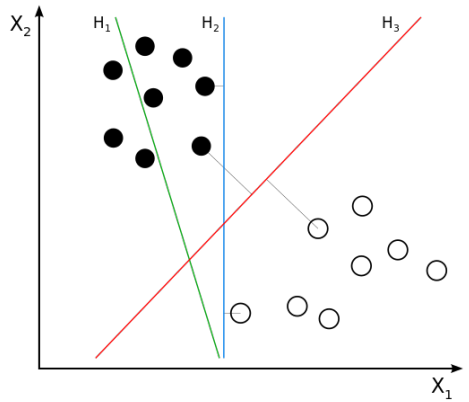
1 Administration

2 Review of Last Lecture

3 Logistic regression

4 Summary

Definition of linear classifiers for binary classification



The decision boundary is a linear function of the input $\boldsymbol{x}$. The prediction is

$$y = \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}) = \begin{cases} +1 & \text{if } \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} > 0 \\ -1 & \text{if } \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \leq 0 \end{cases}$$

Sometimes, we use sgn for sign. Note that a correct classification happens *if and only if*

$$y^* \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} > 0$$

where y^* is the true label.

Perceptron

Iteratively improving one case at a time

- REPEAT
- Pick a data point $\boldsymbol{x}_n$ randomly
- Make a prediction $y = \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$ using the *current* $\boldsymbol{w}$
- If $y = y_n$, do nothing. Else,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y_n \boldsymbol{x}_n$$

- UNTIL converged.

Note that the new $\boldsymbol{w}$ is used to make prediction as soon as it is updated.

Properties

- If the training data is linearly separable, the algorithm stops in a finite number of steps.
- The parameter vector is always a linear combination of training instances.

Stochastic gradient descent for linear regression

RSS Loss function

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n - y_n)^2$$

Widrow-Hoff rule: update parameters using one example at a time

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_n$
  2. Compute its contribution to the gradient (ignoring the constant factor)

$$\boldsymbol{g}_n = (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}}^{(t)} - y_n)\tilde{\boldsymbol{x}}_n$$

  3. Update the parameters $\tilde{\boldsymbol{w}}^{(t+1)} = \tilde{\boldsymbol{w}}^{(t)} - \eta \boldsymbol{g}_n$
  4. $t \leftarrow t + 1$

Outline

1 Administration

2 Review of Last Lecture

3 Logistic regression
- Motivation
- General setup
- Maximum likelihood estimation
- Numerical optimization
- Gradient descent
- Gradient descent for logistic regression
- Newton method
- Stochastic Gradient Descent for Logistic Regression

4 Summary

Linear regression method for classification?

Setup for two classes

- Input: $\boldsymbol{x} \in \mathbb{R}^D$
- Output: $y \in \{0, 1\}$
- Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, N\}$

Why not just use RSS?

$$\tilde{\boldsymbol{w}}_{\text{RSS}} = \arg\min_{\tilde{\boldsymbol{w}}} \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n - y_n)^2$$

Problmes

- $\tilde{\boldsymbol{w}}_{\text{RSS}}^{\mathrm{T}} \tilde{\boldsymbol{x}}$ is not guaranteed to be either 0, 1 or even between 0 and 1!
- Unclear why the binary valued y is appropriate as response variable in regression

Strange thing can happen

Toy data

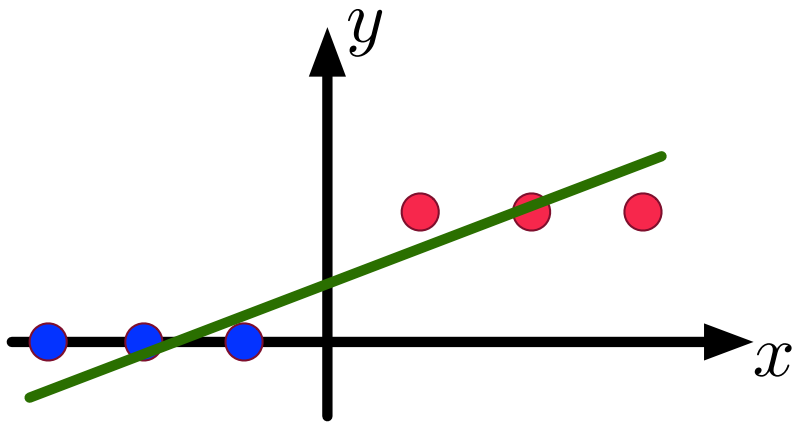
x	label
1	1
2	1
3	1
-1	0
-2	0
-3	0

An intuitive classifier

$$y = \text{sign}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

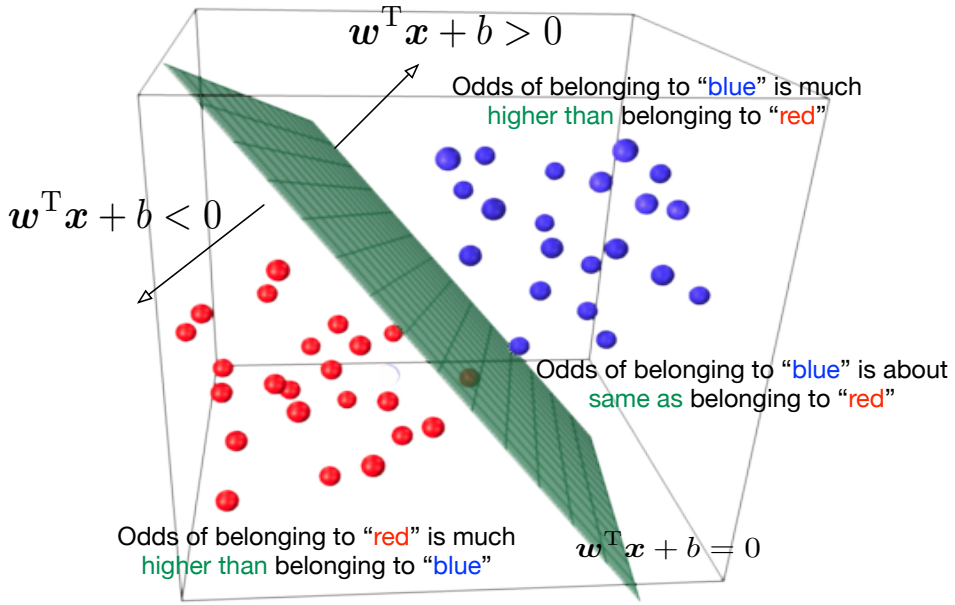
If we use linear regression

We get something not sensible!



Clearly, new perspective is needed!

Reinterpreting the linear decision boundary



The further the point is away from the boundary, the more confident we think it belongs to one of the classes. If a point is on the boundary precisely, then we think it can belong to either class.

We define thus an *odd ratio* to represent how much a point should be interpreted as belonging to class 1 or 0 *over equally possible*.

Link probability to odd ratio

Probability belonging to class 1

$$p(y = 1|\boldsymbol{x})$$

Probability belonging to class 0

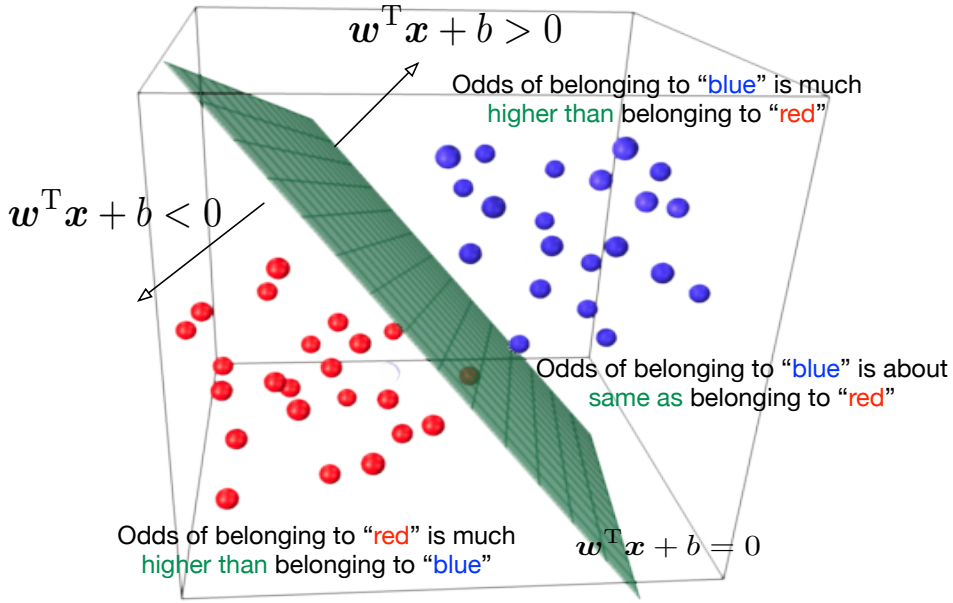
$$p(y = 0|\boldsymbol{x}) = 1 - p(y = 1|\boldsymbol{x})$$

Odd ratio of belonging to class 1

$$r = \frac{p(y = 1|\boldsymbol{x})}{p(y = 0|\boldsymbol{x})} = \frac{p(y = 1|\boldsymbol{x})}{1 - p(y = 1|\boldsymbol{x})}$$

- $r = 1$: equally likely
- $r = +\infty$: absolutely certain class 1
- $r = 0$: absolutely certain class 0

Linking log-odds ratio to decision boundary



$$\log r = \log \frac{p(y=1|\boldsymbol{x})}{1-p(y=1|\boldsymbol{x})} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$$

Or

$$p(y=1|\boldsymbol{x}) = \frac{1}{1+e^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b)}}$$

Logistic classification

Setup for two classes

- Input: $\boldsymbol{x} \in \mathbb{R}^D$
- Output: $y \in \{0, 1\}$
- Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, N\}$
- Model:

$$p(y = 1|\boldsymbol{x}; b, \boldsymbol{w}) = \sigma[g(\boldsymbol{x})]$$

where

$$g(\boldsymbol{x}) = b + \sum_d w_d x_d = b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}$$

and $\sigma[\cdot]$ stands for the *sigmoid* function

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

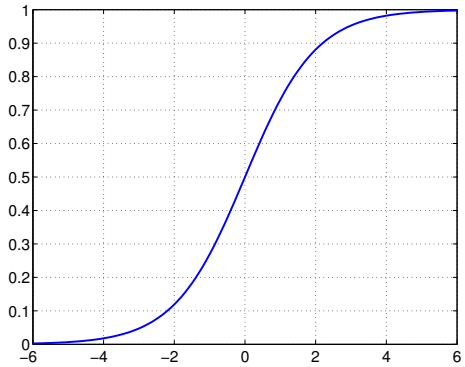
Why the sigmoid function?

What does it look like?

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

where

$$a = b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$$



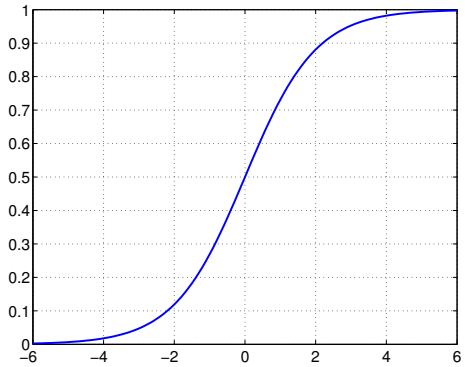
Why the sigmoid function?

What does it look like?

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

where

$$a = b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$$



Properties

- Bounded between 0 and 1 ← thus, interpretable as probability
- Monotonically increasing thus, usable to derive classification rules
  1. $\sigma(a) > 0.5$, positive (classify as '1')
  2. $\sigma(a) < 0.5$, negative (classify as '0')
  3. $\sigma(a) = 0.5$, undecidable
- Nice computationally properties These will unfold in the next few slides

Linear or nonlinear?

$\sigma(a)$ **is nonlinear**, however, the decision boundary is determined by

$$\sigma(a) = 0.5 \Rightarrow a = 0 \Rightarrow g(\boldsymbol{x}) = b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = 0$$

which is a *linear* function in $\boldsymbol{x}$

We often call b the bias term.

Likelihood function

Probability of a single training sample $(\boldsymbol{x}_n, y_n)$

$$p(y_n|\boldsymbol{x}_n; b, \boldsymbol{w}) = \begin{cases} \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) & \text{if} \quad y_n = 1 \\ 1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) & \text{otherwise} \end{cases}$$

Likelihood function

Probability of a single training sample $(\boldsymbol{x}_n, y_n)$

$$p(y_n|\boldsymbol{x}_n; b, \boldsymbol{w}) = \begin{cases} \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) & \text{if} \quad y_n = 1 \\ 1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) & \text{otherwise} \end{cases}$$

Compact expression, exploring that y_n is either 1 or 0

$$p(y_n|\boldsymbol{x}_n; b, \boldsymbol{w}) = \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)^{y_n}[1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]^{1-y_n}$$

Or its logarithm

$$\log p(y_n|\boldsymbol{x}_n; b, \boldsymbol{w}) = \{y_n \log \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\}$$

Cross-entropy error

Log-likelihood of the whole training data $\mathcal{D}$

$$\begin{aligned} \log P(\mathcal{D}) &= \log \prod_n p(y_n|\boldsymbol{x}_n; b, \boldsymbol{w}) \qquad (1) \\ &= \sum_n \{y_n \log \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\} \end{aligned}$$

Cross-entropy error

Log-likelihood of the whole training data $\mathcal{D}$

$$\begin{aligned} \log P(\mathcal{D}) &= \log \prod_n p(y_n|\boldsymbol{x}_n; b, \boldsymbol{w}) \qquad (1) \\ &= \sum_n \{y_n \log \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\} \end{aligned}$$

It is convenient to work with its negation, which is called *cross-entropy error function*

$$\mathcal{E}(b, \boldsymbol{w}) = -\sum_n \{y_n \log \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\}$$

Shorthand notation

This is for convenience

- Append 1 to $\boldsymbol{x}$

$$\boldsymbol{x} \leftarrow [1 \quad x_1 \quad x_2 \quad \cdots \quad x_D]$$

- Append b to $\boldsymbol{w}$

$$\boldsymbol{w} \leftarrow [b \quad w_1 \quad w_2 \quad \cdots \quad w_D]$$

- Cross-entropy is then

$$\mathcal{E}(\boldsymbol{w}) = -\sum_n \{y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)]\}$$

NB. We are not using the $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{w}}$ (as in several textbooks) for cosmetic reasons.

How to find the optimal parameters for logistic regression?

We will minimize the error function

$$\mathcal{E}(\boldsymbol{w}) = -\sum_n \{y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\}$$

However, this function is *highly nonlinear* and we will not be able to find the simple closed-form solution. So we need to use *numerical* methods.

- Numerical methods are messier, in contrast to cleaner analytic solutions.
- In practice, we often have to tune a few optimization parameters — patience is necessary.

An overview of numerical methods

We describe two

- Gradient descent (our focus in lecture): simple, especially effective for large-scale problems
- Newton method: classical and powerful method

Gradient descent is often referred to as a *first-order* method as it requires only to compute the gradients (i.e., the first-order derivative) of the function.

In contrast, Newton method is often referred as to a *second-order* method.

Gradient descent

General form for minimizing $f(\theta)$

$$\boldsymbol{\theta}^{t+1} \leftarrow \boldsymbol{\theta}^{t} - \eta \frac{\partial f}{\partial \boldsymbol{\theta}}$$

Remarks

- η is often called *step size* – literally, how far our update will go along the the direction of the negative gradient
- Note that this is for *minimizing* a function, hence the subtraction $(-\eta)$
- With a *suitable* choice of η, the iterative procedure converges to a stationary point where
$$\frac{\partial f}{\partial \boldsymbol{\theta}} = 0$$
- A stationary point is only necessary for being the minimum.

How do we do this for logistic regression?

Simple fact: derivatives of $\sigma(a)$

$$\frac{d\,\sigma(a)}{d\,a} = \frac{d}{d\,a}\left(\frac{1}{1+e^{-a}}\right) = \frac{-(1+e^{-a})'}{(1+e^{-a})^2}$$

How do we do this for logistic regression?

Simple fact: derivatives of $\sigma(a)$

$$\frac{d\,\sigma(a)}{d\,a} = \frac{d}{d\,a}\left(\frac{1}{1+e^{-a}}\right) = \frac{-(1+e^{-a})'}{(1+e^{-a})^2}$$
$$= \frac{e^{a}}{(1+e^{-a})^2} = \frac{1}{1+e^{-a}}\left(1-\frac{1}{1+e^{-a}}\right)$$

How do we do this for logistic regression?

Simple fact: derivatives of $\sigma(a)$

$$\begin{aligned}\frac{d\,\sigma(a)}{d\,a} &= \frac{d}{d\,a}\left(\frac{1}{1+e^{-a}}\right) = \frac{-(1+e^{-a})'}{(1+e^{-a})^2} \\ &= \frac{e^{a}}{(1+e^{-a})^2} = \frac{1}{1+e^{-a}}\left(1-\frac{1}{1+e^{-a}}\right) \\ &= \sigma(a)[1-\sigma(a)]\end{aligned}$$

Gradients of the cross-entropy error function

Gradients

$$\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}} = -\sum_n \left\{ y_n [1 - \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)] \boldsymbol{x}_n - (1 - y_n) \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)] \boldsymbol{x}_n \right\} \tag{2}$$

$$= \sum_n \left\{ \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) - y_n \right\} \boldsymbol{x}_n \tag{3}$$

Remarks

- $e_n = \left\{ \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) - y_n \right\}$ is called *error* for the nth training sample.
- Stationary point (in this case, the optimum):

$$\sum_n \{ \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) - y_n \} \boldsymbol{x}_n = 0$$

Intuition: *on average, the error is zero*.

Numerical optimization

Gradient descent

- Choose a proper step size $\eta > 0$

Numerical optimization

Gradient descent

- Choose a proper step size $\eta > 0$
- Iteratively update the parameters following the negative gradient to minimize the error function

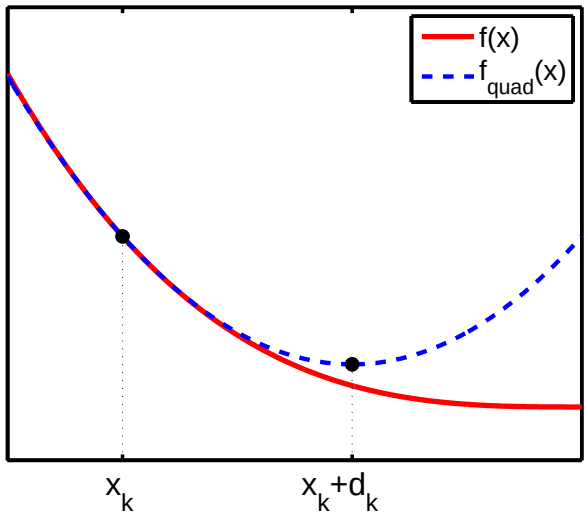
$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \sum_n \left\{ \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) - y_n \right\} \boldsymbol{x}_n$$

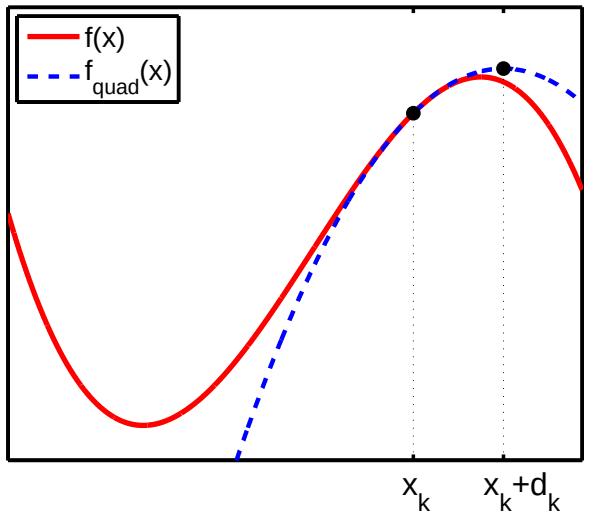
Remarks

- The step size needs to be chosen carefully to ensure convergence.
- The step size can be adaptive (i.e. varying from iteration to iteration). For example, we can use techniques such as *line search*
- There is a variant called *stochastic* gradient descent – we will discuss in a bit.

Intuition for Newton method

Approximate the true function with an easy-to-solve optimization problem





Approximation

Taylor expansion of the cross-entropy function

$$\mathcal{E}(\boldsymbol{w}) \approx \mathcal{E}(\boldsymbol{w}^{(t)}) + (\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \nabla \mathcal{E}(\boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \boldsymbol{H}^{(t)} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

where

- $\nabla \mathcal{E}(\boldsymbol{w}^{(t)})$ is the gradient
- $\boldsymbol{H}^{(t)}$ is the Hessian matrix evaluated at $\boldsymbol{w}^{(t)}$

Example: a scalar function

$$\sin(\theta) \approx \sin(0) + \theta \cos(\theta = 0) + \frac{1}{2}\theta^2[-\sin(\theta = 0)] = \theta$$

where $\nabla \sin(\theta) = \cos(\theta)$ and $\boldsymbol{H} = \nabla \cos(\theta) = -\sin(\theta)$

So what is the Hessian matrix?

The matrix of second-order derivatives

$$\boldsymbol{H} = \frac{\partial^2 \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w} \boldsymbol{w}^{\mathrm{T}}}$$

In other words,

$$H_{ij} = \frac{\partial}{\partial w_j} \left(\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial w_i} \right)$$

So the Hessian matrix is $\mathbb{R}^{\mathsf{D} \times \mathsf{D}}$, where $\boldsymbol{w} \in \mathbb{R}^{\mathsf{D}}$.

Optimizing the approximation

Minimize the approximation

$$\mathcal{E}(\boldsymbol{w}) \approx \mathcal{E}(\boldsymbol{w}^{(t)}) + (\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \nabla \mathcal{E}(\boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \boldsymbol{H}^{(t)} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

and use the solution as the new estimate of the parameters

$$\boldsymbol{w}^{(t+1)} \leftarrow \min_{\boldsymbol{w}} (\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \nabla \mathcal{E}(\boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \boldsymbol{H}^{(t)} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

Optimizing the approximation

Minimize the approximation

$$\mathcal{E}(\boldsymbol{w}) \approx \mathcal{E}(\boldsymbol{w}^{(t)}) + (\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \nabla \mathcal{E}(\boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \boldsymbol{H}^{(t)} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

and use the solution as the new estimate of the parameters

$$\boldsymbol{w}^{(t+1)} \leftarrow \min_{\boldsymbol{w}} (\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \nabla \mathcal{E}(\boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \boldsymbol{H}^{(t)} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

The quadratic function minimization has a *closed* form, thus, we have

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \left(\boldsymbol{H}^{(t)}\right)^{-1} \nabla \mathcal{E}(\boldsymbol{w}^{(t)})$$

i.e., the Newton method.

Contrast gradient descent and Newton method

Similar

Both are iterative procedures.

Difference

- Newton method requires second-order derivatives.
- Newton method does not have the magic η to be set.

Other important things about Hessian

Our cross-entropy error function is convex

$$\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_n \{\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n\}\boldsymbol{x}_n \tag{4}$$

$$\Rightarrow \boldsymbol{H} = \frac{\partial^2 \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}\boldsymbol{w}^{\mathrm{T}}} \tag{5}$$

Other important things about Hessian

Our cross-entropy error function is convex

$$\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_n \{\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n\}\boldsymbol{x}_n \tag{4}$$

$$\Rightarrow \boldsymbol{H} = \frac{\partial^2 \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}\boldsymbol{w}^{\mathrm{T}}} \tag{5}$$

For any vector $\boldsymbol{v}$,

$$\boldsymbol{v}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{v} \geq 0 \geq 0$$

Thus, positive semidefinite. Thus, the cross-entropy error function is convex, with only one global optimum.

Good about Newton method

Fast!

Suppose we want to minimize $f(x) = x^2 + 2x$ and we have its current estimate at $x^{(t)} \neq -1$. So what is the next estimate?

$$x^{(t+1)} \leftarrow x^{(t)} - [f''(x)]^{-1} f'(x) = x^{(t)} - \frac{1}{2}(2x^{(t)} + 2) = -1$$

Namely, the next step (of iteration) immediately tells us the global optimum! (In optimization, this is called *superlinear convergence rate*).

In general, the better our approximation, the fast the Newton method is in solving our optimization problem.

Bad about Newton method

Not scalable!

- Computing and inverting Hessian matrix can be very expensive for large-scale problems where the dimensionally D is very large.
- Newton method does not guarantee convergence if your starting point is far away from the optimum

NB. There are fixes and alternatives, such as Quasi-Newton/Quasi-second order method.

Stochastic gradient descent

Cross entropy error function

$$\mathcal{E}(\boldsymbol{w}) = -\sum_n \{y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\}$$

Update rules

- Initialize $\boldsymbol{w}$ to $\boldsymbol{w}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$

Stochastic gradient descent

Cross entropy error function

$$\mathcal{E}(\boldsymbol{w}) = -\sum_n \{y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1-y_n)\log[1-\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\}$$

Update rules

- Initialize $\boldsymbol{w}$ to $\boldsymbol{w}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_n$
  2. Compute its contribution to the gradient

$$\boldsymbol{g}_n = (\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n)\boldsymbol{x}_n$$

Stochastic gradient descent

Cross entropy error function

$$\mathcal{E}(\boldsymbol{w}) = -\sum_n \{y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1-y_n)\log[1-\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\}$$

Update rules

- Initialize $\boldsymbol{w}$ to $\boldsymbol{w}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_n$
  2. Compute its contribution to the gradient

$$\boldsymbol{g}_n = (\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n)\boldsymbol{x}_n$$

  3. Update the parameters $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \boldsymbol{g}_n$

Stochastic gradient descent

Cross entropy error function

$$\mathcal{E}(\boldsymbol{w}) = -\sum_n \{y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1-y_n)\log[1-\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\}$$

Update rules

- Initialize $\boldsymbol{w}$ to $\boldsymbol{w}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_n$
  2. Compute its contribution to the gradient

$$\boldsymbol{g}_n = (\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n)\boldsymbol{x}_n$$

  3. Update the parameters $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \boldsymbol{g}_n$
  4. $t \leftarrow t+1$

So why $(\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n)$ is called error?

Recall $\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$ is the probability

$$p(y = 1|\boldsymbol{x}_n)$$

Given the update rule,

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \boldsymbol{g}_n = \boldsymbol{w}^{(t)} - \eta(\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n)\boldsymbol{x}_n$$

The effect is:

- If $y_n = 1$ and $\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) \approx 1$, we do nothing
- If $y_n = 1$ and $\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) \approx 0$, we *add* $\boldsymbol{x}_n$ to $\boldsymbol{w}^{(t)}$
- If $y_n = 0$ and $\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) \approx 1$, we *subtract* $\boldsymbol{x}_n$ to $\boldsymbol{w}^{(t)}$
- If $y_n = 0$ and $\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) \approx 0$, we do nothing

This rule is similar to perceptron update rule except "softer" as $\sigma(\cdot)$ is often between 0 and 1

Outline

1 Administration

2 Review of Last Lecture

3 Logistic regression

4 Summary

Important concepts and tools

- Logistic regression:
  probabilistic modeling of the (log) odds ratio
  cross-entropy error function
- Numerical optimization:
  First-order method: gradient descent, batch update, stochastic gradient descent
  Second-order method: Newton method