Logistic Regression

Professor Ameet Talwalkar

Outline

1. Administration
2. Review of last lecture
3. Logistic regression

HW1

- Will be returned today in class during our break
- Can also pick up from Brooke during her office hours

Outline

How to tell spam from ham?

FROM THE DESK OF MR. AMINU SALEH
DIRECTOR, FOREIGN OPERATIONS DEPARTMENT
AFRI BANK PLC
Afribank Plaza,
14th Floormoney344.jpg
51/55 Broad Street,
P.M.B 12021 Lagos-Nigeria

Attention: Honorable Beneficiary,

IMMEDIATE PAYMENT NOTIFICATION VALUED AT **US$10 MILLION**

Dear Ameet,

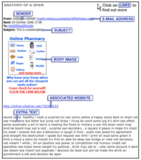
Do you have 10 minutes to get on a videocall before 2pm?

Thanks,

Stefano

Simple strategy: count the words

Bag-of-word representation
of documents (and textual data)



$$\begin{pmatrix} \text{free} & 100 \\ \text{money} & 2 \\ \vdots & \vdots \\ \text{account} & 2 \\ \vdots & \vdots \end{pmatrix}$$

Just wanted to send a quick reminder about the guest lect
noon. We meet in RTH 105. It has a PC and LCD projector
connection for your laptop if you desire. Maybe we can
to setup the A/V stuff.

Again, if you would be able to make it around 30 minutes
great.

Thanks so much for your willingness to do this,
Mark

$$\begin{pmatrix} \text{free} & 1 \\ \text{money} & 1 \\ \vdots & \vdots \\ \text{account} & 2 \\ \vdots & \vdots \end{pmatrix}$$

Naive Bayes (in our Spam Email Setting)

Assume $X \in \mathbb{R}^{\mathsf{D}}$, all $X_d \in [\mathsf{K}]$, and z_k is the number of times k in X

$$P(X = x, Y = c) = P(Y = c)P(X = x|Y = c)$$

Naive Bayes (in our Spam Email Setting)

Assume $X \in \mathbb{R}^{\mathsf{D}}$, all $X_d \in [\mathsf{K}]$, and z_k is the number of times k in X

$$\begin{aligned} P(X = x, Y = c) &= P(Y = c)P(X = x|Y = c) \\ &= P(Y = c)\prod_k P(k|Y = c)^{z_k} = \pi_c \prod_k \theta_{ck}^{z_k} \end{aligned}$$

Key assumptions made?

Naive Bayes (in our Spam Email Setting)

Assume $X \in \mathbb{R}^{\mathsf{D}}$, all $X_d \in [\mathsf{K}]$, and z_k is the number of times k in X

$$\begin{aligned} P(X=x, Y=c) &= P(Y=c)P(X=x|Y=c) \\ &= P(Y=c)\prod_k P(k|Y=c)^{z_k} = \pi_c \prod_k \theta_{ck}^{z_k} \end{aligned}$$

Key assumptions made?

- Conditional independence: $P(X_i, X_j|Y=c) = P(X_i|Y=c)P(X_j|Y=c)$.
- $P(X_i|Y=c)$ depends only the value of X_i, not i itself (order of words does not matter in "bag-of-word" representation of documents)

Learning problem

Training data

$$\mathcal{D} = \{(\{z_{nk}\}_{k=1}^{\mathsf{K}}, y_n)\}_{n=1}^{\mathsf{N}}$$

Goal

Learning problem

Training data

$$\mathcal{D} = \{(\{z_{nk}\}_{k=1}^{\mathsf{K}}, y_n)\}_{n=1}^{\mathsf{N}}$$

Goal

Learn $\pi_c, c = 1, 2, \cdots, \mathsf{C}$, and $\theta_{ck}, \forall c \in [\mathsf{C}], k \in [\mathsf{K}]$ under the constraints:

Learning problem

Training data

$$\mathcal{D} = \{(\{z_{nk}\}_{k=1}^{\mathsf{K}}, y_n)\}_{n=1}^{\mathsf{N}}$$

Goal

Learn $\pi_c, c = 1, 2, \cdots, \mathsf{C}$, and $\theta_{ck}, \forall c \in [\mathsf{C}], k \in [\mathsf{K}]$ under the constraints:

$$\sum_c \pi_c = 1\,,$$

$$\sum_k \theta_{ck} = \sum_k P(k|Y = c) = 1\,,$$

and all quantities should be nonnegative.

Likelihood Function

- Let $X_1, \ldots, X_N$ be IID with PDF $f(x|\theta)$ (also written as $f(x;\theta)$)
- *Likelihood function* is defined by $L(\theta|x)$ (also written as $L(\theta;x)$):

$$L(\theta|x) = \prod_{i=1}^{N} f(X_i;\theta).$$

Notes The likelihood function is just the joint density of the data, except that we treat it as a function of the parameter θ, $L : \Theta \to [0,\infty)$.

Maximum Likelihood Estimator

Definition: The maximum likelihood estimator (MLE) $\hat{\theta}$, is the value of θ that maximizes $L(\theta)$.

The log-likelihood function is defined by $l(\theta) = \log L(\theta)$

- Maximum occurs at same place as that of the likelihood function

Maximum Likelihood Estimator

Definition: The maximum likelihood estimator (MLE) $\hat{\theta}$, is the value of θ that maximizes $L(\theta)$.

The log-likelihood function is defined by $l(\theta) = \log L(\theta)$

- Maximum occurs at same place as that of the likelihood function
- Using logs simplifies mathematical expressions (converts exponents to products and products to sums)
- Using logs helps with numerical stabilitity

The same is true of the likelihood function times any constant. Thus we shall often drop constants in the likelihood function.

Bayes Rule

For any document x, we want to compare $p(\text{spam}|x)$ and $p(\text{ham}|x)$

Axiom of Probability: $p(\text{spam}, x) = p(\text{spam}|x)p(x) = p(x|\text{spam})p(\text{spam})$

This gives us (via bayes rule):

$$p(\text{spam}|x) = \frac{p(x|\text{spam})p(\text{spam})}{p(x)}$$

$$p(\text{ham}|x) = \frac{p(x|\text{ham})p(\text{ham})}{p(x)}$$

Bayes Rule

For any document x, we want to compare $p(\text{spam}|x)$ and $p(\text{ham}|x)$

Axiom of Probability: $p(\text{spam}, x) = p(\text{spam}|x)p(x) = p(x|\text{spam})p(\text{spam})$

This gives us (via bayes rule):

$$p(\text{spam}|x) = \frac{p(x|\text{spam})p(\text{spam})}{p(x)}$$

$$p(\text{ham}|x) = \frac{p(x|\text{ham})p(\text{ham})}{p(x)}$$

Denominators are same, and easier to compute logarithms, so we compare:

$$\log[p(x|\text{spam})p(\text{spam})] \quad \text{versus} \quad \log[p(x|\text{ham})p(\text{ham})]$$

Our hammer: maximum likelihood estimation

Log-Likelihood of the training data

$$\mathcal{L} = \log P(\mathcal{D}) = \log \prod_{n=1}^{\mathsf{N}} \pi_{y_n} P(x_n | y_n)$$

Our hammer: maximum likelihood estimation

Log-Likelihood of the training data

$$\mathcal{L} = \log P(\mathcal{D}) = \log \prod_{n=1}^{\mathsf{N}} \pi_{y_n} P(x_n|y_n)$$

$$= \log \prod_{n=1}^{\mathsf{N}} \left(\pi_{y_n} \prod_{k} \theta_{y_n k}^{z_{nk}} \right)$$

Our hammer: maximum likelihood estimation

Log-Likelihood of the training data

$$
\begin{aligned}
\mathcal{L} = \log P(\mathcal{D}) &= \log \prod_{n=1}^{\mathsf{N}} \pi_{y_n} P(x_n|y_n) \\
&= \log \prod_{n=1}^{\mathsf{N}} \left(\pi_{y_n} \prod_k \theta_{y_n k}^{z_{nk}} \right) \\
&= \sum_n \left(\log \pi_{y_n} + \sum_k z_{nk} \log \theta_{y_n k} \right)
\end{aligned}
$$

Our hammer: maximum likelihood estimation

Log-Likelihood of the training data

$$
\begin{aligned}
\mathcal{L} = \log P(\mathcal{D}) &= \log \prod_{n=1}^{\mathsf{N}} \pi_{y_n} P(x_n|y_n) \\
&= \log \prod_{n=1}^{\mathsf{N}} \left(\pi_{y_n} \prod_k \theta_{y_n k}^{z_{nk}} \right) \\
&= \sum_n \left(\log \pi_{y_n} + \sum_k z_{nk} \log \theta_{y_n k} \right) \\
&= \sum_n \log \pi_{y_n} + \sum_{n,k} z_{nk} \log \theta_{y_n k}
\end{aligned}
$$

Our hammer: maximum likelihood estimation

Log-Likelihood of the training data

$$
\begin{aligned}
\mathcal{L} = \log P(\mathcal{D}) &= \log \prod_{n=1}^{\mathsf{N}} \pi_{y_n} P(x_n | y_n) \\
&= \log \prod_{n=1}^{\mathsf{N}} \left(\pi_{y_n} \prod_k \theta_{y_n k}^{z_{nk}} \right) \\
&= \sum_n \left(\log \pi_{y_n} + \sum_k z_{nk} \log \theta_{y_n k} \right) \\
&= \sum_n \log \pi_{y_n} + \sum_{n,k} z_{nk} \log \theta_{y_n k}
\end{aligned}
$$

Optimize it!

$$
(\pi_c^*, \theta_{ck}^*) = \arg\max \sum_n \log \pi_{y_n} + \sum_{n,k} z_{nk} \log \theta_{y_n k}
$$

Details

Note the separation of parameters in the likelihood

$$\sum_n \log \pi_{y_n} + \sum_{n,k} z_{nk} \log \theta_{y_n k}$$

which implies that $\{\pi_c\}$ and $\{\theta_{ck}\}$ can be estimated separately.

Reorganize terms

$$\sum_n \log \pi_{y_n} = \sum_c \log \pi_c \times (\#\text{of data points labeled as c})$$

Details

Note the separation of parameters in the likelihood

$$\sum_n \log \pi_{y_n} + \sum_{n,k} z_{nk} \log \theta_{y_n k}$$

which implies that $\{\pi_c\}$ and $\{\theta_{ck}\}$ can be estimated separately.

Reorganize terms

$$\sum_n \log \pi_{y_n} = \sum_c \log \pi_c \times (\#\text{of data points labeled as c})$$

and

$$\sum_{n,k} z_{nk} \log \theta_{y_n k} = \sum_c \sum_{n:y_n=c} \sum_k z_{nk} \log \theta_{ck} = \sum_c \sum_{n:y_n=c,k} z_{nk} \log \theta_{ck}$$

The later implies $\{\theta_{ck}, k = 1, 2, \cdots, \mathsf{K}\}$ and $\{\theta_{c'k}, k = 1, 2, \cdots, \mathsf{K}\}$ can be estimated independently.

Estimating $\{\pi_c\}$

We want to maximize

$$\sum_c \log \pi_c \times (\#\text{of data points labeled as c})$$

Intuition

- Similar to roll a dice (or flip a coin): each side of the dice shows up with a probability of π_c (total C sides)
- And we have total N trials of rolling this dice

Solution

Estimating $\{\pi_c\}$

We want to maximize

$$\sum_c \log \pi_c \times (\text{\#of data points labeled as c})$$

Intuition

- Similar to roll a dice (or flip a coin): each side of the dice shows up with a probability of π_c (total C sides)
- And we have total N trials of rolling this dice

Solution

$$\pi_c^* = \frac{\text{\#of data points labeled as c}}{\text{N}}$$

Estimating $\{\theta_{ck}, k = 1, 2, \cdots, \mathsf{K}\}$

We want to maximize

$$\sum_{n:y_n=c,k} z_{nk} \log \theta_{ck}$$

Intuition

- Again similar to roll a dice: each side of the dice shows up with a probability of θ_{ck} (total K sides)
- And we have total $\sum_{n:y_n=c,k} z_{nk}$ trials.

Solution

Estimating $\{\theta_{ck}, k = 1, 2, \cdots, \mathsf{K}\}$

We want to maximize

$$\sum_{n:y_n=c,k} z_{nk} \log \theta_{ck}$$

Intuition

- Again similar to roll a dice: each side of the dice shows up with a probability of θ_{ck} (total K sides)
- And we have total $\sum_{n:y_n=c,k} z_{nk}$ trials.

Solution

$$\theta^*_{ck} = \frac{\text{\#of times side k shows up in data points labeled as c}}{\text{\#total trials for data points labeled as c}}$$

Translating back to our problem of detecting spam emails

- Collect a lot of ham and spam emails as training examples
- Estimate the "bias"

$$p(\text{ham}) = \frac{\#\text{of ham emails}}{\#\text{of emails}}, \quad p(\text{spam}) = \frac{\#\text{of spam emails}}{\#\text{of emails}}$$

- Estimate the weights (i.e., $p(\text{funny\_word}|\text{ham})$ etc)

$$p(\text{funny\_word}|\text{ham}) = \frac{\#\text{of funny\_word in ham emails}}{\#\text{of words in ham emails}} \tag{1}$$

$$p(\text{funny\_word}|\text{spam}) = \frac{\#\text{of funny\_word in spam emails}}{\#\text{of words in spam emails}} \tag{2}$$

Classification rule

Given an unlabeled data point $x = \{z_k, k = 1, 2, \cdots, \mathsf{K}\}$, label it with

$$
\begin{aligned}
y^* &= \arg\max_{c \in [\mathsf{C}]} P(y = c|x) \\
&= \arg\max_{c \in [\mathsf{C}]} P(y = c) P(x|y = c) \\
&= \arg\max_c [\log \pi_c + \sum_k z_k \log \theta_{ck}]
\end{aligned}
$$

Constrained optimization

Equality Constraints

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g(x) = 0 \end{aligned}$$

Method of Lagrange multipliers

Construct the following function (Lagrangian)

$$L(x, \lambda) = f(x) + \lambda g(x)$$

A short derivation of the maximum likelihood estimation

To maximize

$$\sum_{n:y_n=c,k} z_{nk} \log \theta_{ck}$$

We can use Lagrange multipliers

A short derivation of the maximum likelihood estimation

To maximize

$$\sum_{n:y_n=c,k} z_{nk} \log \theta_{ck}$$

We can use Lagrange multipliers

$$f(\theta) = -\sum_{n:y_n=c,k} z_{nk} \log \theta_{ck}$$

$$g(\theta) = 1 - \sum_k \theta_{ck} = 0$$

Lagrangian

$$\begin{aligned} L(\theta, \lambda) &= f(\theta) + \lambda g(\theta) \\ &= -\sum_{n:y_n=c,k} z_{nk} \log \theta_{ck} + \lambda \left(1 - \sum_k \theta_{ck}\right) \end{aligned}$$

$$L(\theta, \lambda) = -\sum_{n:y_n=c,k} z_{nk} \log \theta_{ck} + \lambda \left(1 - \sum_k \theta_{ck}\right)$$

First take derivatives with respect to θ_{ck} and then find the stationary point

$$-\left(\sum_{n:y_n=c} \frac{z_{nk}}{\theta_{ck}}\right) - \lambda = 0 \rightarrow \theta_{ck} = -\frac{1}{\lambda} \sum_{n:y_n=c} z_{nk}$$

$$L(\theta, \lambda) = -\sum_{n:y_n=c,k} z_{nk} \log \theta_{ck} + \lambda \left(1 - \sum_k \theta_{ck}\right)$$

First take derivatives with respect to θ_{ck} and then find the stationary point

$$-\left(\sum_{n:y_n=c} \frac{z_{nk}}{\theta_{ck}}\right) - \lambda = 0 \rightarrow \theta_{ck} = -\frac{1}{\lambda} \sum_{n:y_n=c} z_{nk}$$

- Plug in expression above for θ_{ck} into constraint $\sum_k \theta_{ck} = 1$
- Solve for λ
- Plug this expression for λ back into expression for θ_{ck} to get:

$$\theta_{ck} = \frac{\sum_{n:y_n=c} z_{nk}}{\sum_k \sum_{n:y_n=c} z_{nk}}$$

Chris will review in section on Friday

Summary

Things you should know

- The form of the naive Bayes model
  - write down the joint distribution
  - explain the conditional independence assumption implied by the model
  - explain how this model can be used to classify spam vs ham emails
  - explain how it could be used for categorical variables
- Be able to go through the short derivation for parameter estimation
  - The model illustrated here is called discrete Naive Bayes
  - HW2 asks you to apply the same principle to other variants of naive Bayes
  - The derivations are very similar – except there you need to estimate different model parameters

Moving forward

Examine the classification rule for naive Bayes

$$y^* = \arg\max_c \log \pi_c + \sum_k z_k \log \theta_{ck}$$

For binary classification, we thus determine the label based on the sign of

$$\log \pi_1 + \sum_k z_k \log \theta_{1k} - \left(\log \pi_2 + \sum_k z_k \log \theta_{2k} \right)$$

This is just a linear function of the features $\{z_k\}$

Moving forward

Examine the classification rule for naive Bayes

$$y^* = \arg\max_c \log \pi_c + \sum_k z_k \log \theta_{ck}$$

For binary classification, we thus determine the label based on the sign of

$$\log \pi_1 + \sum_k z_k \log \theta_{1k} - \left(\log \pi_2 + \sum_k z_k \log \theta_{2k} \right)$$

This is just a linear function of the features $\{z_k\}$

$$w_0 + \sum_k z_k w_k$$

where we "absorb" $w_0 = \log \pi_1 - \log \pi_2$ and $w_k = \log \theta_{1k} - \log \theta_{2k}$.

Naive Bayes is a linear classifier

Fundamentally, what really matters in deciding decision boundary is

$$w_0 + \sum_k z_k w_k$$

This motivates many new methods, including logistic regression, to be discussed next

Outline

Logistic classification

Setup for two classes

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

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?

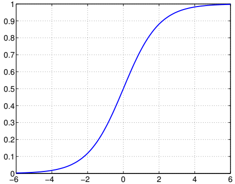
Why the sigmoid function?

What does it look like?

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

where

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



Properties

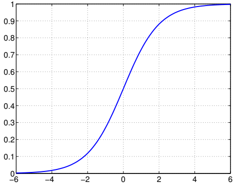
Why the sigmoid function?

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 $\leftarrow$ thus, interpretable as probability
- Monotonically increasing thus, usable to derive classification rules
  - $\sigma(a) > 0.5$, positive (classify as '1')
  - $\sigma(a) < 0.5$, negative (classify as '0')
  - $\sigma(a) = 0.5$, undecidable
- Nice computational properties As we will see soon

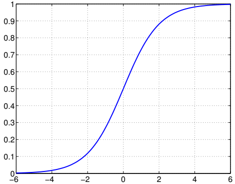
Why the sigmoid function?

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 $\leftarrow$ thus, interpretable as probability
- Monotonically increasing thus, usable to derive classification rules
  - $\sigma(a) > 0.5$, positive (classify as '1')
  - $\sigma(a) < 0.5$, negative (classify as '0')
  - $\sigma(a) = 0.5$, undecidable
- Nice computational properties As we will see soon

Linear or nonlinear classifier?

Linear or nonlinear?

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

$$\sigma(a) = 0.5 \Rightarrow$$

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 offset term.

Contrast Naive Bayes and our new model

Similar

Contrast Naive Bayes and our new model

Similar

Both classification models are linear functions of features

Different

Contrast Naive Bayes and our new model

Similar

Both classification models are linear functions of features

Different

Naive Bayes models the *joint* distribution: $P(X, Y) = P(Y)P(X|Y)$
Logistic regression models the *conditional* distribution: $P(Y|X)$

Generative vs. Discriminative

NB is a *generative* model, LR is a *discriminative* model

- We will talk more about the differences later

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**

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

Log Likelihood or Cross Entropy Error

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

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

Log Likelihood or Cross Entropy Error

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

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

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

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

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 complex and we cannot find the simple solution as we did in Naive Bayes. 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's method: classical and powerful method

Gradient descent is often referred to as a *first-order* method

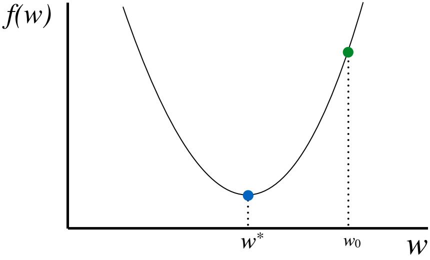
- Requires computation of gradients (i.e., the first-order derivative)

Newton's method is often referred as to an *second-order* method

- Requires computation of second derivatives

Gradient Descent

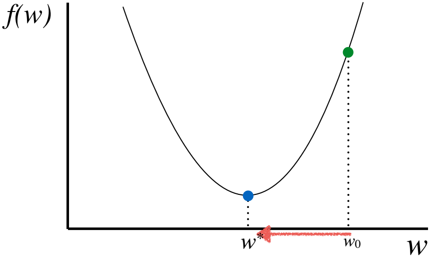
Start at a random point



Gradient Descent

Start at a random point

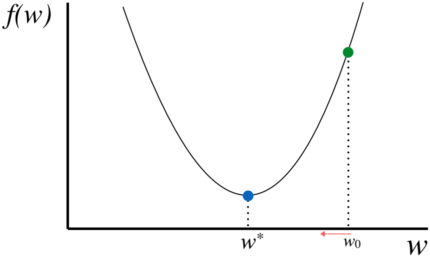
Determine a descent direction



Gradient Descent

Start at a random point

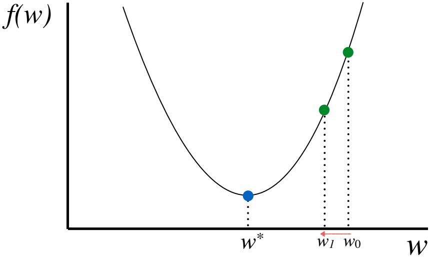
Determine a descent direction
Choose a step size



Gradient Descent

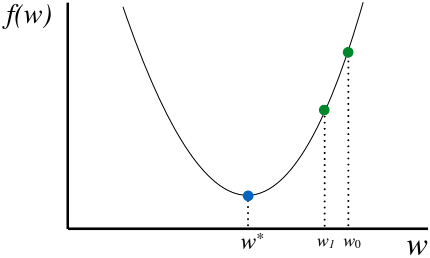
Start at a random point

- Determine a descent direction
- Choose a step size
- Update



Gradient Descent

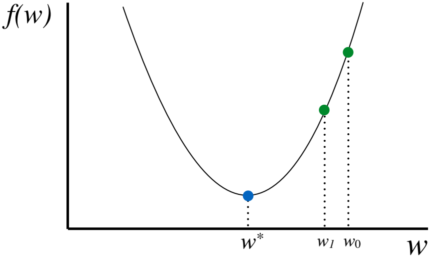
Start at a random point
Repeat
  Determine a descent direction
  Choose a step size
  Update
Until stopping criterion is satisfied



Gradient Descent

Start at a random point
Repeat
- Determine a descent direction
- Choose a step size
- Update

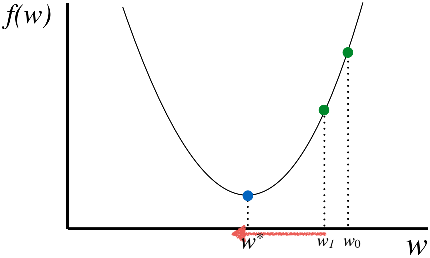
Until stopping criterion is satisfied



Gradient Descent

Start at a random point
Repeat
- Determine a descent direction
- Choose a step size
- Update

Until stopping criterion is satisfied



Gradient Descent

Start at a random point

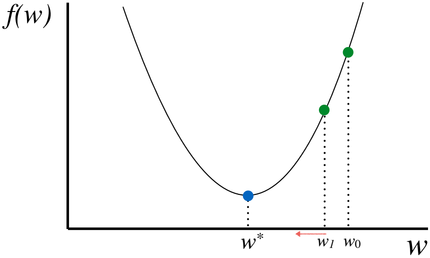
Repeat

Determine a descent direction

Choose a step size

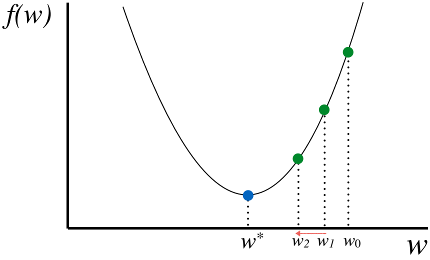
Update

Until stopping criterion is satisfied



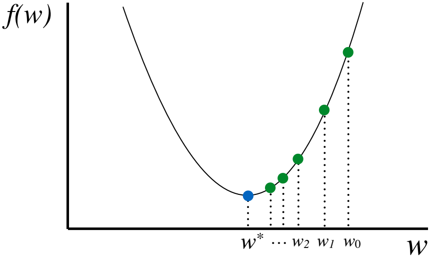
Gradient Descent

Start at a random point
Repeat
Determine a descent direction
Choose a step size
Update
Until stopping criterion is satisfied

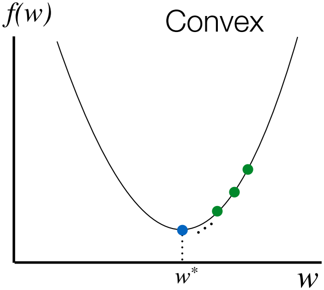


Gradient Descent

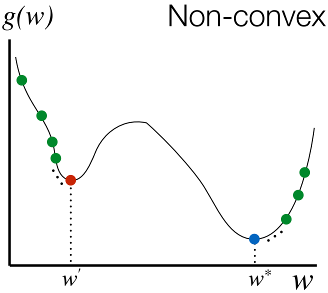
Start at a random point
Repeat
    Determine a descent direction
    Choose a step size
    Update
Until stopping criterion is satisfied



Where Will We Converge?



Any local minimum is a global minimum

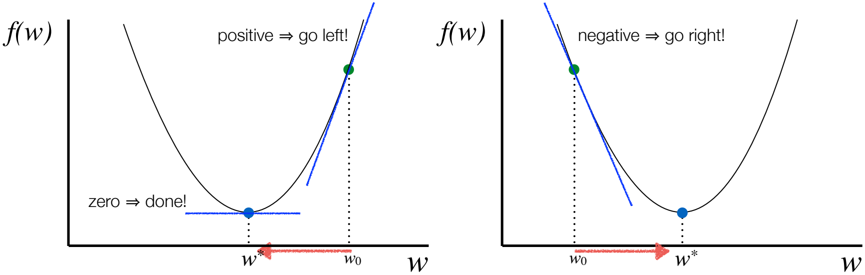


Multiple local minima may exist

Least Squares, Ridge Regression and Logistic Regression are all convex!

Why do we move in the direction opposite the gradient?

Choosing Descent Direction (1D)



We can only move in two directions

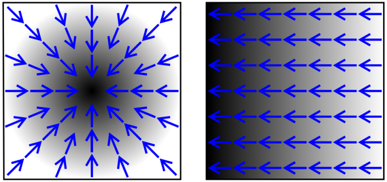
Negative slope is direction of descent!

Step Size

Update Rule: $w_{i+1} = w_i - \alpha_i \frac{df}{dw}(w_i)$

Negative Slope

Choosing Descent Direction



"Gradient2" by Sarang. Licensed under CC BY-SA 2.5 via Wikimedia Commons
http://commons.wikimedia.org/wiki/File:Gradient2.svg#/media/File:Gradient2.svg

2D Example:

- Function values are in black/white and black represents higher values
- Arrows are gradients

We can move anywhere in $\mathbb{R}^d$

Negative gradient is direction of *steepest* descent!

Step Size

Update Rule: $\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \nabla f(\mathbf{w}_i)$

Negative Slope

Example: $\min f(\boldsymbol{\theta}) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2$

Example: $\min f(\boldsymbol{\theta}) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2$

- We compute the gradients

$$\frac{\partial f}{\partial \theta_1} = 2(\theta_1^2 - \theta_2)\theta_1 + \theta_1 - 1$$
$$\frac{\partial f}{\partial \theta_2} = -(\theta_1^2 - \theta_2)$$

Example: $\min f(\boldsymbol{\theta}) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2$

- We compute the gradients

$$\frac{\partial f}{\partial \theta_1} = 2(\theta_1^2 - \theta_2)\theta_1 + \theta_1 - 1$$
$$\frac{\partial f}{\partial \theta_2} = -(\theta_1^2 - \theta_2)$$

- Use the following *iterative* procedure for *gradient descent*
  1. Initialize $\theta_1^{(0)}$ and $\theta_2^{(0)}$, and $t = 0$
  2. do

$$\theta_1^{(t+1)} \leftarrow \theta_1^{(t)} - \eta \left[2(\theta_1^{(t)}{}^2 - \theta_2^{(t)})\theta_1^{(t)} + \theta_1^{(t)} - 1 \right]$$
$$\theta_2^{(t+1)} \leftarrow \theta_2^{(t)} - \eta \left[-(\theta_1^{(t)}{}^2 - \theta_2^{(t)}) \right]$$
$$t \leftarrow t + 1$$

  3. until $f(\boldsymbol{\theta}^{(t)})$ *does not change much*

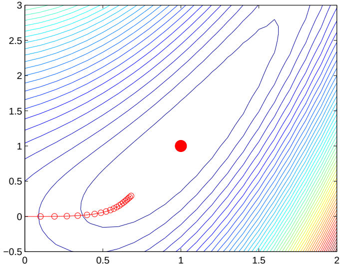
Impact of step size

Choosing the right η is important

Impact of step size

Choosing the right η is important

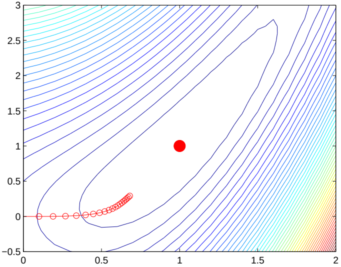
small η is too slow?



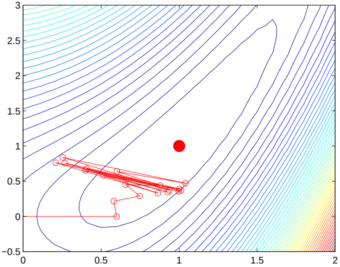
Impact of step size

Choosing the right η is important

small η is too slow?



large η is too unstable?



Gradient descent

General form for minimizing $f(\boldsymbol{\theta})$

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

Remarks

- η is *step size* – how far we go in the direction of the negative gradient
  - Step size needs to be chosen carefully to ensure convergence.
  - Step size can be adaptive, e.g., we can use *line search*
- We are *minimizing* a function, hence the subtraction $(-\eta)$
- With a *suitable* choice of η, we converge to a stationary point

$$\frac{\partial f}{\partial \boldsymbol{\theta}} = 0$$

- Stationary point not always global minimum (but happy when convex)
- Popular variant called *stochastic* gradient descent

Gradient Descent Update for Logistic Regression

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

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

Gradients of the cross-entropy error function

Cross-entropy Error Function

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

Gradients

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

Remark

Gradients of the cross-entropy error function

Cross-entropy Error Function

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

Gradients

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

Remark

- $e_n = \left\{\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n\right\}$ is called *error* for the nth training sample.

Numerical optimization

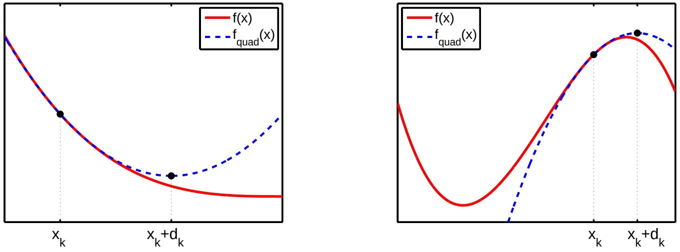
Gradient descent for logistic regression

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

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

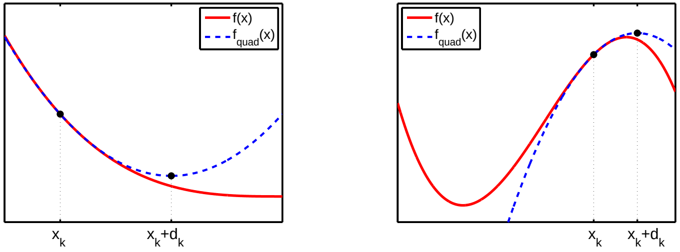
Intuition for Newton's method

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



Intuition for Newton's method

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



In particular, we can approximate the cross-entropy error function around $\boldsymbol{w}^{(t)}$ by a quadratic function, and then minimize this quadratic function

Approximation

Second Order Taylor expansion around x_t

$$f(x) \approx f(x_t) + f'(x_t)(x - x_t) + \frac{1}{2}f''(x_t)(x - x_t)^2$$

Approximation

Second Order Taylor expansion around x_t

$$f(x) \approx f(x_t) + f'(x_t)(x - x_t) + \frac{1}{2} f''(x_t)(x - x_t)^2$$

Taylor expansion of cross-entropy error function around $\boldsymbol{w}^{(t)}$

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

where

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

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's method.

Contrast gradient descent and Newton's method

Similar

- Both are iterative procedures.

Different

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

Other important things about Hessian

Our cross-entropy error function is convex

$$\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_n \{\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n\}\boldsymbol{x}_n \tag{3}$$

$$\Rightarrow \boldsymbol{H} = \frac{\partial^2 \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}\boldsymbol{w}^{\mathrm{T}}} = \text{homework} \tag{4}$$

Other important things about Hessian

Our cross-entropy error function is convex

$$\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_n \{\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n\}\boldsymbol{x}_n \tag{3}$$

$$\Rightarrow \boldsymbol{H} = \frac{\partial^2 \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}\boldsymbol{w}^{\mathrm{T}}} = \text{homework} \tag{4}$$

For any vector $\boldsymbol{v}$,

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

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

Good about Newton's method

Fast (in terms of convergence)!

Newton's method finds the optimal point in a *single* iteration when the function we're optimizing is quadratic

In general, the better our Taylor approximation, the more quickly Newton's method will converge

Bad about Newton's method

Not scalable!

Computing and inverting Hessian matrix can be very expensive for large-scale problems where the dimensionally D is very large. *There are fixes and alternatives, such as Quasi-Newton/Quasi-second order method.*

Summary

Setup for 2 classes

- Logistic Regression models conditional distribution as: $p(y=1|\boldsymbol{x};\boldsymbol{w}) = \sigma[g(\boldsymbol{x})]$ where $g(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$
- Linear decision boundary: $g(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = 0$

Minimizing cross-entropy error (negative log-likelihood)

- $\mathcal{E}(b,\boldsymbol{w}) = -\sum_n \{y_n \log \sigma(b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1-y_n)\log[1-\sigma(b+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\}$
- No closed form solution; must rely on iterative solvers

Numerical optimization

- Gradient descent: simple, scalable to large-scale problems
  - move in direction opposite of gradient!
  - gradient of logistic function takes nice form
- Newton method: fast to converge but not scalable
  - At each iteration, find optimal point in 2nd-order Taylor expansion
  - Closed form solution exists for each iteration

Naive Bayes and logistic regression: two different modeling paradigms

- Maximize *joint* likelihood $\sum_n \log p(\boldsymbol{x}_n, y_n)$ (Generative, NB)
- Maximize *conditional* likelihood $\sum_n \log p(y_n|\boldsymbol{x}_n)$ (Discriminative, LR)
- More on this next class