Drs. Sha & Liu

\{feisha, yanliu.cs\}@usc.edu

September 22, 2014
Outline

1. Administration
2. Logistic Regression - continued
3. Multiclass classification
A few announcements

- Homework 1: due 9/24 (see the homework sheets for detailed submission information)
- Revised lecture slides are on Blackboard and the course website
Outline

1 Administration

2 Logistic Regression - continued
   - Logistic regression
   - Numerical optimization
   - Gradient descent
   - Gradient descent for logistic regression
   - Newton method

3 Multiclass classification
Logistic classification

Setup for two classes

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

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

where

\[
g(\mathbf{x}) = b + \sum_d w_d x_d = b + \mathbf{w}^T \mathbf{x}
\]

- Linear decision boundary

\[
g(\mathbf{x}) = b + \mathbf{w}^T \mathbf{x} = 0
\]
Maximum likelihood estimation

Cross-entropy error (negative log-likelihood)

\[ \mathcal{E}(b, w) = - \sum_{n} \{ y_n \log \sigma(b + w^T x_n) + (1 - y_n) \log[1 - \sigma(b + w^T x_n)] \} \]

Numerical optimization

- Gradient descent: simple, scalable to large-scale problems
- Newton method: fast but not scalable
Logistic Regression - continued

Shorthand notation

This is for convenience

- Append 1 to $\mathbf{x}$
  \[ \mathbf{x} \leftarrow [1 \ x_1 \ x_2 \ \cdots \ x_D] \]
- Append $b$ to $\mathbf{w}$
  \[ \mathbf{w} \leftarrow [b \ w_1 \ w_2 \ \cdots \ w_D] \]
- Cross-entropy is then
  \[ \mathcal{E}(\mathbf{w}) = - \sum_{n} \{ y_n \log \sigma(\mathbf{w}^T \mathbf{x}_n) + (1 - y_n) \log[1 - \sigma(\mathbf{w}^T \mathbf{x}_n)] \} \]

NB. We are not using the $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{w}}$ (as in several textbooks) for cosmetic reasons.
How to find the optimal parameters for logistic regression?

We will minimize the error function

\[ E(w) = -\sum_n \{y_n \log \sigma(w^T x_n) + (1 - y_n) \log[1 - \sigma(w^T x_n)]\} \]

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 method: classical and powerful method

Gradient descent is often referred to as an 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 an second-order method.
Example: \( \min f(\theta) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2 \)

- We compute the gradients

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

\[
\frac{\partial f}{\partial \theta_2} = -(\theta_1^2 - \theta_2) \quad (2)
\]

- Use the following iterative procedure for gradient descent

1. Initialize \( \theta_1^{(0)} \) and \( \theta_2^{(0)} \), and \( t = 0 \)
2. do

\[
\theta_1^{(t+1)} \leftarrow \theta_1^{(t)} - \eta \left[ 2(\theta_1^{(t)^2} - \theta_2^{(t)})\theta_1^{(t)} + \theta_1^{(t)} - 1 \right] \quad (3)
\]

\[
\theta_2^{(t+1)} \leftarrow \theta_2^{(t)} - \eta \left[ -(\theta_1^{(t)^2} - \theta_2^{(t)}) \right] \quad (4)
\]

\[
t \leftarrow t + 1 \quad (5)
\]
3. until \( f(\theta^{(t)}) \) does not change much
Gradient descent

General form for minimizing $f(\theta)$

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

Remarks

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

Choose the right $\eta$ is important

small $\eta$ is too slow?
Seeing in action

Choose the right $\eta$ is important

small $\eta$ is too slow?

large $\eta$ is too unstable?
How do we do this for logistic regression?

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

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

\[
\frac{d \sigma(a)}{da} = \frac{d}{da} \left( \frac{1}{1 + e^{-a}} \right) = \frac{-(1 + e^{-a})'}{(1 + e^{-a})^2}
\]

\[
= \frac{e^a}{(1 + e^{-a})^2} = \frac{1}{1 + e^{-a}} \left( 1 - \frac{1}{1 + e^{-a}} \right)
\]

\[
= \sigma(a) [1 - \sigma(a)]
\]

\[
\frac{d \log \sigma(a)}{da} = 1 - \sigma(a)
\]
Gradients of the cross-entropy error function

**Gradients**

\[
\frac{\partial E(w)}{\partial w} = - \sum_n \{ y_n [1 - \sigma(w^T x_n)] x_n - (1 - y_n) \sigma(w^T x_n) x_n \} \tag{6}
\]

\[
= \sum_n \{ \sigma(w^T x_n) - y_n \} x_n \tag{7}
\]

**Remarks**

- \( e_n = \{ \sigma(w^T x_n) - y_n \} \) is called error for the \( n \)th training sample.
- Stationary point (in this case, the optimum):
  \[
  \sum_n \sigma(w^T x_n) x_n = \sum_n x_n y_n
  \]

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

$$w(t+1) \leftarrow w(t) - \eta \sum_{n} \left\{ \sigma(w^T x_n) - y_n \right\} x_n$$

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, also popularly used (later in this semester).
Intuition for Newton method

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

\[ f(x) \approx f_{quad}(x) \]
Approximation

**Taylor expansion of the cross-entropy function**

\[ \mathcal{E}(\mathbf{w}) \approx \mathcal{E}(\mathbf{w}^{(t)}) + (\mathbf{w} - \mathbf{w}^{(t)})^T \nabla \mathcal{E}(\mathbf{w}^{(t)}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^{(t)})^T \mathbf{H}^{(t)}(\mathbf{w} - \mathbf{w}^{(t)}) \]

where

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

**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 \( \mathbf{H} = \nabla \cos(\theta) = - \sin(\theta) \)
So what is the Hessian matrix?

The matrix of second-order derivatives

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

In other words,

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

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

Minimize the approximation

\[ \mathcal{E}(w) \approx \mathcal{E}(w^{(t)}) + (w - w^{(t)})^T \nabla \mathcal{E}(w^{(t)}) + \frac{1}{2} (w - w^{(t)})^T H^{(t)} (w - w^{(t)}) \]

and use the solution as the new estimate of the parameters

\[ w^{(t+1)} \leftarrow \min_w (w - w^{(t)})^T \nabla \mathcal{E}(w^{(t)}) + \frac{1}{2} (w - w^{(t)})^T H^{(t)} (w - w^{(t)}) \]
Optimizing the approximation

Minimize the approximation

\[
E(w) \approx E(w(t)) + (w - w(t))^T \nabla E(w(t)) + \frac{1}{2} (w - w(t))^T H(t) (w - w(t))
\]

and use the solution as the new estimate of the parameters

\[
w^{(t+1)} \leftarrow \min_w (w - w(t))^T \nabla E(w(t)) + \frac{1}{2} (w - w(t))^T H(t) (w - w(t))
\]

The quadratic function minimization has a \textit{closed} form, thus, we have

\[
w^{(t+1)} \leftarrow w(t) - \left( H(t) \right)^{-1} \nabla E(w(t))
\]

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 $\eta$ to be set.
Our cross-entropy error function is convex

\[
\frac{\partial \mathcal{E}(w)}{\partial w} = \sum_n \{ \sigma(w^T x_n) - y_n \} x_n
\]  

\[\Rightarrow H = \frac{\partial^2 \mathcal{E}(w)}{\partial w w^T} = \text{homework}\]
Our cross-entropy error function is convex

\[
\frac{\partial \mathcal{E}(w)}{\partial w} = \sum_n \{ \sigma(w^T x_n) - y_n \} x_n
\]  

\[\Rightarrow H = \frac{\partial^2 \mathcal{E}(w)}{\partial w w^T} = \text{homework} \]

For any vector \(v\),

\[v^T H v = \text{homework} \geq 0\]

Thus, positive definite. 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?

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

Namely, the next step (of iteration) immediately tells us the global optimum! (In optimization, this is called \textit{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.
Outline

1. Administration

2. Logistic Regression - continued

3. Multiclass classification
   - Use binary classifiers as building blocks
   - Multinomial logistic regression
Suppose we need to predict multiple classes/outcomes:
$C_1, C_2, \ldots, C_K$

- Weather prediction: sunny, cloudy, raining, etc
- Optical character recognition: 10 digits + 26 characters (lower and upper cases) + special characters, etc

**Studied methods**

- Nearest neighbor classifier
- Naive Bayes
- Gaussian discriminant analysis
- Logistic regression
Logistic regression for predicting multiple classes? Easy

The approach of “one versus the rest”

- For each class $C_k$, change the problem into binary classification
  1. Relabel training data with label $C_k$, into POSITIVE (or ‘1’)
  2. Relabel all the rest data into NEGATIVE (or ‘0’)

This step is often called 1-of-$K$ encoding. That is, only one is nonzero and everything else is zero.

Example: for class $C_2$, data go through the following change

$$(x_1, C_1) \rightarrow (x_1, 0), (x_2, C_3) \rightarrow (x_2, 0), \ldots, (x_n, C_2) \rightarrow (x_n, 1), \ldots,$$
Logistic regression for predicting multiple classes? Easy

The approach of “one versus the rest”

- For each class $C_k$, change the problem into binary classification
  1. Relabel training data with label $C_k$, into POSITIVE (or ‘1’)
  2. Relabel all the rest data into NEGATIVE (or ‘0’)

This step is often called 1-of-$K$ encoding. That is, only one is nonzero and everything else is zero.

Example: for class $C_2$, data go through the following change

$$(x_1, C_1) \rightarrow (x_1, 0), (x_2, C_3) \rightarrow (x_2, 0), \ldots, (x_n, C_2) \rightarrow (x_n, 1), \ldots,$$

- Train $K$ binary classifiers using logistic regression to differentiate the two classes
Logistic regression for predicting multiple classes? Easy

The approach of “one versus the rest”
- For each class $C_k$, change the problem into binary classification
  1. Relabel training data with label $C_k$, into POSITIVE (or ‘1’)
  2. Relabel all the rest data into NEGATIVE (or ‘0’)

This step is often called 1-of-$K$ encoding. That is, only one is nonzero and everything else is zero.

Example: for class $C_2$, data go through the following change

\[(x_1, C_1) \rightarrow (x_1, 0), (x_2, C_3) \rightarrow (x_2, 0), \ldots, (x_n, C_2) \rightarrow (x_n, 1), \ldots,\]

- Train $K$ binary classifiers using logistic regression to differentiate the two classes
- When predicting on $x$, combine the outputs of all binary classifiers
  1. What if all the classifiers say NEGATIVE?
Logistic regression for predicting multiple classes? Easy

The approach of “one versus the rest”

- For each class $C_k$, change the problem into binary classification
  1. Relabel training data with label $C_k$, into POSITIVE (or ‘1’)
  2. Relabel all the rest data into NEGATIVE (or ‘0’)

This step is often called 1-of-$K$ encoding. That is, only one is nonzero and everything else is zero.

Example: for class $C_2$, data go through the following change

$$(x_1, C_1) \rightarrow (x_1, 0), (x_2, C_3) \rightarrow (x_2, 0), \ldots, (x_n, C_2) \rightarrow (x_n, 1), \ldots,$$

- Train $K$ binary classifiers using logistic regression to differentiate the two classes
- When predicting on $x$, combine the outputs of all binary classifiers
  1. What if all the classifiers say NEGATIVE?
  2. What if multiple classifiers say POSITIVE?

Take-home exercise: there are different combination strategies. Can you think of any?
Yet, another easy approach

The approach of “one versus one”

- For each pair of classes $C_k$ and $C_{k'}$, change the problem into binary classification
  1. Relabel training data with label $C_k$, into POSITIVE (or ‘1’)
  2. Relabel training data with label $C_{k'}$ into NEGATIVE (or ‘0’)
  3. *Disregard* all other data

Ex: for class $C_1$ and $C_2$,

$$(x_1, C_1), (x_2, C_3), (x_3, C_2), \ldots \rightarrow (x_1, 1), (x_3, 0), \ldots$$
Yet, another easy approach

The approach of “one versus one”

- For each pair of classes $C_k$ and $C_{k'}$, change the problem into binary classification
  1. Relabel training data with label $C_k$, into positive (or ‘1’)
  2. Relabel training data with label $C_{k'}$ into negative (or ‘0’)
  3. Disregard all other data

Ex: for class $C_1$ and $C_2$,

$$(x_1, C_1), (x_2, C_3), (x_3, C_2), \ldots \rightarrow (x_1, 1), (x_3, 0), \ldots$$

- Train $K(K - 1)/2$ binary classifiers using logistic regression to differentiate the two classes
Multiclass classification

Use binary classifiers as building blocks

Yet, another easy approach

The approach of “one versus one”

- For each *pair* of classes $C_k$ and $C_{k'}$, change the problem into binary classification
  1. Relabel training data with label $C_k$, into **POSITIVE** (or ‘1’)
  2. Relabel training data with label $C_{k'}$ into **NEGATIVE** (or ‘0’)
  3. *Disregard* all other data

Ex: for class $C_1$ and $C_2$,

$$(x_1, C_1), (x_2, C_3), (x_3, C_2), \ldots \rightarrow (x_1, 1), (x_3, 0), \ldots$$

- Train $K(K-1)/2$ binary classifiers using logistic regression to differentiate the two classes
- When predicting on $x$, combine the outputs of all binary classifiers
  There are $K(K-1)/2$ votes! **Take-home exercise:** *can you think of any good combination strategies?*
Contrast these two approaches

Pros and cons of each approach

- **one versus the rest**: only needs to train $K$ classifiers. Make a huge difference if you have a lot of classes to go through. Can you think of a good application example where there are a lot of classes?
Contrast these two approaches

Pros and cons of each approach

- **one versus the rest**: only needs to train $K$ classifiers. Make a huge difference if you have a lot of *classes* to go through. Can you think of a good application example where there are a lot of classes?

- **one versus one**: only needs to train a smaller subset of data (only those labeled with those two classes would be involved). Make a huge difference if you have a lot of *data* to go through.
Contrast these two approaches

Pros and cons of each approach

- **one versus the rest**: only needs to train \( K \) classifiers. Make a huge difference if you have a lot of classes to go through. Can you think of a good application example where there are a lot of classes?

- **one versus one**: only needs to train a smaller subset of data (only those labeled with those two classes would be involved). Make a huge difference if you have a lot of data to go through.

Bad about both of them

*Combining classifiers’ outputs seem to be a bit tricky.* Any other good methods?
Multinomial logistic regression

Intuition: from the decision rule of our naive Bayes classifier

\[ y^* = \arg \max_c p(y = c \mid \mathbf{x}) = \arg \max_c \log p(x \mid y = c) p(y = c) \]
\[ = \arg \max_c \log \pi_c + \sum_k z_k \log \theta_{ck} = \arg \max_c \mathbf{w}_c^T \mathbf{x} \]

Essentially, we are comparing

\[ \mathbf{w}_1^T \mathbf{x}, \mathbf{w}_2^T \mathbf{x}, \cdots, \mathbf{w}_C^T \mathbf{x} \]

with one for each category.
So, can we define the following conditional model?

\[ p(y = c | \mathbf{x}) = \sigma\left[ \mathbf{w}_c^T \mathbf{x} \right] \]
So, can we define the following conditional model?

\[ p(y = c|x) = \sigma [w_c^T x] \]

This would not work at least for the reason

\[ \sum_c p(y = c|x) = \sum_c \sigma [w_c^T x] \neq 1 \]

as each the summand can be any number (independently) between 0 and 1. But we are close
Definition of multinomial logistic regression

**Model**

For each class $C_k$, we have a parameter vector $\mathbf{w}_k$ and model the posterior probability as

$$p(C_k|x) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{k'} e^{\mathbf{w}_{k'}^T \mathbf{x}}} \quad \leftarrow \quad \text{This is called softmax function}$$
Definition of multinomial logistic regression

Model

For each class $C_k$, we have a parameter vector $\mathbf{w}_k$ and model the posterior probability as

$$p(C_k|\mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{k'} e^{\mathbf{w}_{k'}^T \mathbf{x}}} \quad \text{← This is called softmax function}$$

Decision boundary: assign $\mathbf{x}$ with the label that is the maximum of posterior

$$\arg \max_k P(C_k|\mathbf{x}) \rightarrow \arg \max_k \mathbf{w}_k^T \mathbf{x}$$

Note: the notation is changed to denote the classes as $C_k$ instead of just $c$