

CS545: Classification with Logistic Regression

Chuck Anderson

Department of Computer Science
Colorado State University

Fall, 2009

Outline

CS545:
Classification with
Logistic Regression

Chuck Anderson

Logistic Regression

Why?

Setup

Derivation

Logistic Regression

Why?

Setup

Derivation

Masking

- Recall that a linear model used for classification can result in masking.

Masking

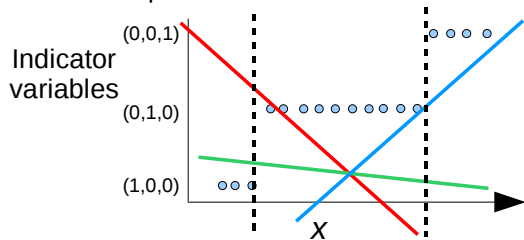
- Recall that a linear model used for classification can result in masking.
- We discussed fixing this by using different shaped membership functions, other than linear.

- Recall that a linear model used for classification can result in masking.
- We discussed fixing this by using different shaped membership functions, other than linear.
- Our first approach to this was to use generative models (Gaussian distributions) to model the data from each class.

- Recall that a linear model used for classification can result in masking.
- We discussed fixing this by using different shaped membership functions, other than linear.
- Our first approach to this was to use generative models (Gaussian distributions) to model the data from each class.
- Using Bayes Theorem, we derived QDA and LDA.

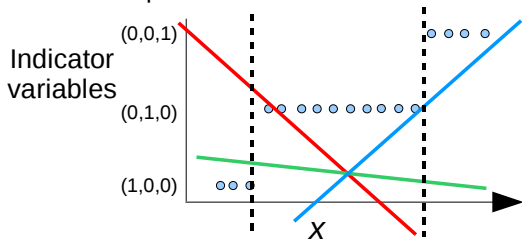
Masking

- Remember this picture?



Masking

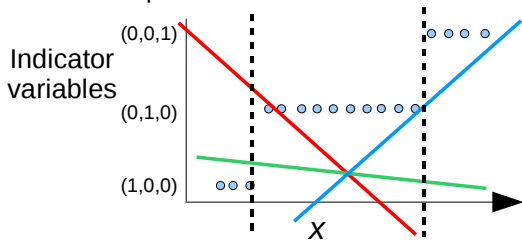
- Remember this picture?



- The problem was that the green line for Class 2 was too low. In fact, all lines are too low in the middle of x range. Maybe we can reduce the masking effect by

Masking

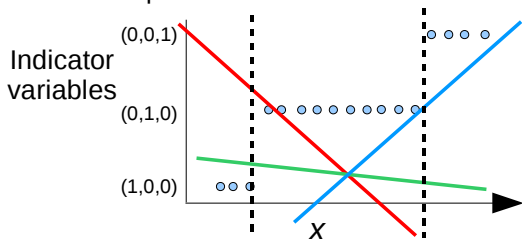
- Remember this picture?



- The problem was that the green line for Class 2 was too low. In fact, all lines are too low in the middle of x range. Maybe we can reduce the masking effect by
 - requiring the function values to be between 0 and 1, and

Masking

- Remember this picture?



- The problem was that the green line for Class 2 was too low. In fact, all lines are too low in the middle of x range. Maybe we can reduce the masking effect by
 - requiring the function values to be between 0 and 1, and
 - requiring them to sum to 1 for every value of x .

Logistic Regression Setup

- We can satisfy those two requirements by directly representing $p(C = k|\mathbf{x})$ as

$$p(C = k|\mathbf{x}) = \frac{f(\mathbf{x}, \beta_k)}{\sum_{m=1}^K f(\mathbf{x}, \beta_m)}$$

where we haven't discussed the form of f yet, but β represents the parameters of f that we will tune to fit the training data (later).

Logistic Regression Setup

- We can satisfy those two requirements by directly representing $p(C = k|\mathbf{x})$ as

$$p(C = k|\mathbf{x}) = \frac{f(\mathbf{x}, \beta_k)}{\sum_{m=1}^K f(\mathbf{x}, \beta_m)}$$

where we haven't discussed the form of f yet, but β represents the parameters of f that we will tune to fit the training data (later).

- This is certainly an expression that is between 0 and 1 for any \mathbf{x} .

Logistic Regression Setup

- We can satisfy those two requirements by directly representing $p(C = k|\mathbf{x})$ as

$$p(C = k|\mathbf{x}) = \frac{f(\mathbf{x}, \beta_k)}{\sum_{m=1}^K f(\mathbf{x}, \beta_m)}$$

where we haven't discussed the form of f yet, but β represents the parameters of f that we will tune to fit the training data (later).

- This is certainly an expression that is between 0 and 1 for any \mathbf{x} .
- Now we have $p(C = k|\mathbf{x})$ expressed directly, as opposed to the previous generative approach of first modeling $p(\mathbf{x}|C = k)$ and using Bayes' theorem to get $p(C = k|\mathbf{x})$.

- Let's give the above expression another name

$$g(\mathbf{x}, \beta_k) = p(C = k|\mathbf{x}) = \frac{f(\mathbf{x}, \beta_k)}{\sum_{m=1}^K f(\mathbf{x}, \beta_m)}$$

- Let's give the above expression another name

$$g(\mathbf{x}, \beta_k) = p(C = k|\mathbf{x}) = \frac{f(\mathbf{x}, \beta_k)}{\sum_{m=1}^K f(\mathbf{x}, \beta_m)}$$

- Now let's deal with our requirement that the sum must equal 1

$$1 = \sum_{k=1}^K p_k(C = k|\mathbf{x}) = \sum_{k=1}^K g(\mathbf{x}, \beta_k)$$

- Let's give the above expression another name

$$g(\mathbf{x}, \beta_k) = p(C = k|\mathbf{x}) = \frac{f(\mathbf{x}, \beta_k)}{\sum_{m=1}^K f(\mathbf{x}, \beta_m)}$$

- Now let's deal with our requirement that the sum must equal 1

$$1 = \sum_{k=1}^K p_k(C = k|\mathbf{x}) = \sum_{k=1}^K g(\mathbf{x}, \beta_k)$$

- However, this constraint overdetermines the $g(\mathbf{x}, \beta_k)$. If

$$1 = a + b + c$$

must be true, then given values for a and b , c is already determined, as $c = 1 - a - b$.

- Let's give the above expression another name

$$g(\mathbf{x}, \beta_k) = p(C = k|\mathbf{x}) = \frac{f(\mathbf{x}, \beta_k)}{\sum_{m=1}^K f(\mathbf{x}, \beta_m)}$$

- Now let's deal with our requirement that the sum must equal 1

$$1 = \sum_{k=1}^K p_k(C = k|\mathbf{x}) = \sum_{k=1}^K g(\mathbf{x}, \beta_k)$$

- However, this constraint overdetermines the $g(\mathbf{x}, \beta_k)$. If

$$1 = a + b + c$$

must be true, then given values for a and b , c is already determined, as $c = 1 - a - b$.

- Another way to say this is that we can set c to any value, and values for a and b can still be found that satisfy the above equation. For example

$$1 = (a - c/2) + (b - c/2) + 2c$$

- So, let's just set the final $f(\mathbf{x}, \beta_k)$ to be 1. Now

$$g(\mathbf{x}, \beta_k) = \begin{cases} \frac{f(\mathbf{x}, \beta_k)}{1 + \sum_{m=1}^{K-1} f(\mathbf{x}, \beta_m)}, & k < K \\ \frac{1}{1 + \sum_{m=1}^{K-1} f(\mathbf{x}, \beta_m)}, & k = K \end{cases}$$

Derivation

- Whatever we choose for f , we must make a plan for optimizing its parameters β . How?

- Whatever we choose for f , we must make a plan for optimizing its parameters β . How?
- Let's maximize the likelihood of the data. So, what is the likelihood of training data consisting of samples $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ and class indicator variables

$$\begin{pmatrix} t_{1,1} & t_{1,2} & \dots & t_{1,K} \\ t_{2,1} & t_{2,2} & \dots & t_{2,K} \\ \vdots & & & \\ t_{N,1} & t_{N,2} & \dots & t_{N,K} \end{pmatrix}$$

with every value $t_{n,k}$ being 0 or 1, and each row of this matrix contains a single 1. (We can also express $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ as an $N \times D$ matrix, but we will be using single samples \mathbf{x}_n more often in the following.)

Data Likelihood

- The likelihood is just the product of all $p(C = \text{class of } n^{\text{th}} \text{ sample} | \mathbf{x}_n)$ values for sample n . A common way to express this product, using those handy indicator variables is

$$L(\beta) = \prod_{n=1}^N \prod_{k=1}^{K-1} p(C = k | \mathbf{x}_n)^{t_{n,k}}$$

Data Likelihood

- The likelihood is just the product of all $p(C = \text{class of } n^{\text{th}} \text{ sample} | \mathbf{x}_n)$ values for sample n . A common way to express this product, using those handy indicator variables is

$$L(\beta) = \prod_{n=1}^N \prod_{k=1}^{K-1} p(C = k | \mathbf{x}_n)^{t_{n,k}}$$

- Why does second product stop with $K - 1$?

Data Likelihood

- The likelihood is just the product of all $p(C = \text{class of } n^{\text{th}} \text{ sample} | \mathbf{x}_n)$ values for sample n . A common way to express this product, using those handy indicator variables is

$$L(\beta) = \prod_{n=1}^N \prod_{k=1}^{K-1} p(C = k | \mathbf{x}_n)^{t_{n,k}}$$

- Why does second product stop with $K - 1$?
- Say we have three classes ($K = 3$) and training sample n is from Class 2, then the inner product is

$$\begin{aligned} p(C = 1 | \mathbf{x}_n)^{t_{n,1}} p(C = 2 | \mathbf{x}_n)^{t_{n,2}} p(C = 3 | \mathbf{x}_n)^{t_{n,3}} \\ &= p(C = 1 | \mathbf{x}_n)^0 p(C = 2 | \mathbf{x}_n)^1 p(C = 3 | \mathbf{x}_n)^0 \\ &= 1 p(C = 2 | \mathbf{x}_n)^1 1 \\ &= p(C = 2 | \mathbf{x}_n) \end{aligned}$$

This shows how the indicator variables as exponents select the correct terms to be included in the product.

Maximizing the Data Likelihood

- So, we want to find β that maximizes the data likelihood. How shall we proceed?

$$L(\beta) = \prod_{n=1}^N \prod_{k=1}^{K-1} p(C = k | \mathbf{x}_n)^{t_{n,k}}$$

Maximizing the Data Likelihood

- So, we want to find β that maximizes the data likelihood. How shall we proceed?

$$L(\beta) = \prod_{n=1}^N \prod_{k=1}^{K-1} p(C = k | \mathbf{x}_n)^{t_{n,k}}$$

- Right. Find the derivative with respect to each component of β , or the gradient with respect to β . But there is a mess of products in this. So...

Maximizing the Data Likelihood

- So, we want to find β that maximizes the data likelihood. How shall we proceed?

$$L(\beta) = \prod_{n=1}^N \prod_{k=1}^{K-1} p(C = k | \mathbf{x}_n)^{t_{n,k}}$$

- Right. Find the derivative with respect to each component of β , or the gradient with respect to β . But there is a mess of products in this. So...
- Right. Work with the logarithm $\log L(\beta)$ which we will call $l(\beta)$.

$$l(\beta) = \log L(\beta) = \sum_{n=1}^N \sum_{k=1}^{K-1} t_{n,k} \log p(C = k | \mathbf{x}_n)$$

Gradient Descent

- Unfortunately, the gradient of $l(\beta)$ with respect to β is not linear in β , so we cannot simply set the result equal to zero and solve for β .

Gradient Descent

- Unfortunately, the gradient of $l(\beta)$ with respect to β is not linear in β , so we cannot simply set the result equal to zero and solve for β .
- Instead, we do gradient descent:

$$\beta \leftarrow \beta + \alpha \nabla_{\beta} l(\beta)$$

where α is a constant that affects the step size.

Gradient Descent

- Unfortunately, the gradient of $l(\beta)$ with respect to β is not linear in β , so we cannot simply set the result equal to zero and solve for β .
- Instead, we do gradient descent:
 - Initialize β to some value.

$$\beta \leftarrow \beta + \alpha \nabla_{\beta} l(\beta)$$

where α is a constant that affects the step size.

Gradient Descent

- Unfortunately, the gradient of $l(\beta)$ with respect to β is not linear in β , so we cannot simply set the result equal to zero and solve for β .
- Instead, we do gradient descent:
 - Initialize β to some value.
 - Make small change to β in the direction of the gradient of $l(\beta)$ with respect to β (or $\nabla_{\beta}l(\beta)$)

$$\beta \leftarrow \beta + \alpha \nabla_{\beta} l(\beta)$$

where α is a constant that affects the step size.

- Unfortunately, the gradient of $l(\beta)$ with respect to β is not linear in β , so we cannot simply set the result equal to zero and solve for β .
- Instead, we do gradient descent:
 - Initialize β to some value.
 - Make small change to β in the direction of the gradient of $l(\beta)$ with respect to β (or $\nabla_{\beta}l(\beta)$)
 - Repeat above step until $l(\beta)$ seems to be at a maximum.

$$\beta \leftarrow \beta + \alpha \nabla_{\beta} l(\beta)$$

where α is a constant that affects the step size.

- Remember that β is a matrix of parameters, with, let's say, columns corresponding to the values required for each f , of which there are $K - 1$.

- Remember that β is a matrix of parameters, with, let's say, columns corresponding to the values required for each f , of which there are $K - 1$.
- We can work on the update formula and $\nabla_{\beta} l(\beta)$ one column at a time

$$\beta_k \leftarrow \beta_k + \alpha \nabla_{\beta_k} l(\beta)$$

and combine them at the end.

$$\beta \leftarrow \beta + \alpha (\nabla_{\beta_1} l(\beta), \nabla_{\beta_2} l(\beta), \dots, \nabla_{\beta_{K-1}} l(\beta))$$

- Remembering that $\frac{\partial \log h(x)}{\partial x} = \frac{1}{h(x)} \frac{\partial h(x)}{x}$ and that $p(C = k | \mathbf{x}_n) = g(\mathbf{x}_n, \beta_k)$

$$l(\beta) = \sum_{n=1}^N \sum_{k=1}^{K-1} t_{n,k} \log p(C = k | \mathbf{x}_n)$$

$$= \sum_{n=1}^N \sum_{k=1}^{K-1} t_{n,k} \log g(\mathbf{x}_n, \beta_k)$$

$$\nabla_{\beta_j} l(\beta) = \sum_{n=1}^N \sum_{k=1}^{K-1} \frac{t_{n,k}}{g(\mathbf{x}_n, \beta_k)} \nabla_{\beta_j} g(\mathbf{x}_n, \beta_k)$$

- Remembering that $\frac{\partial \log h(x)}{\partial x} = \frac{1}{h(x)} \frac{\partial h(x)}{x}$ and that $p(C = k | \mathbf{x}_n) = g(\mathbf{x}_n, \beta_k)$

$$l(\beta) = \sum_{n=1}^N \sum_{k=1}^{K-1} t_{n,k} \log p(C = k | \mathbf{x}_n)$$

$$= \sum_{n=1}^N \sum_{k=1}^{K-1} t_{n,k} \log g(\mathbf{x}_n, \beta_k)$$

$$\nabla_{\beta_j} l(\beta) = \sum_{n=1}^N \sum_{k=1}^{K-1} \frac{t_{n,k}}{g(\mathbf{x}_n, \beta_k)} \nabla_{\beta_j} g(\mathbf{x}_n, \beta_k)$$

- It would be super nice if $\nabla_{\beta_j} g(\mathbf{x}_n, \beta_k)$ includes the factor $g(\mathbf{x}_n, \beta_k)$ so that it will cancel with the $g(\mathbf{x}_n, \beta_k)$ in the denominator.

Chuck Anderson

Logistic Regression
Why?
Setup
Derivation

- Can get this by defining

$$f(\mathbf{x}_n, \beta_k) = e^{\beta_k^T \mathbf{x}_n} \quad \text{so}$$
$$g(\mathbf{x}_n, \beta_k) = \frac{e^{\beta_k^T \mathbf{x}_n}}{1 + \sum_{m=1}^{K-1} e^{\beta_m^T \mathbf{x}_n}}$$

- Can get this by defining

$$f(\mathbf{x}_n, \beta_k) = e^{\beta_k^T \mathbf{x}_n} \quad \text{so}$$

$$g(\mathbf{x}_n, \beta_k) = \frac{e^{\beta_k^T \mathbf{x}_n}}{1 + \sum_{m=1}^{K-1} e^{\beta_m^T \mathbf{x}_n}}$$

- Now we can work on $\nabla_{\beta_j} g(\mathbf{x}_n, \beta_k)$

$$\begin{aligned} &= \nabla_{\beta_j} \left(1 + \sum_{m=1}^{K-1} e^{\beta_m^T \mathbf{x}_n} \right)^{-1} e^{\beta_k^T \mathbf{x}_n} \\ &= -1 \left(1 + \sum_{m=1}^{K-1} e^{\beta_m^T \mathbf{x}_n} \right)^{-2} e^{\beta_j^T \mathbf{x}_n} \mathbf{x}_n e^{\beta_k^T \mathbf{x}_n} + \left(1 + \sum_{m=1}^{K-1} e^{\beta_m^T \mathbf{x}_n} \right)^{-1} e^{\beta_j^T \mathbf{x}_n} \mathbf{x}_n \\ &= -\frac{e^{\beta_k^T \mathbf{x}_n}}{1 + \sum_{m=1}^{K-1} e^{\beta_m^T \mathbf{x}_n}} \frac{e^{\beta_j^T \mathbf{x}_n}}{1 + \sum_{m=1}^{K-1} e^{\beta_j^T \mathbf{x}_n}} \mathbf{x}_n + \frac{e^{\beta_j^T \mathbf{x}_n}}{1 + \sum_{m=1}^{K-1} e^{\beta_j^T \mathbf{x}_n}} \mathbf{x}_n \\ &= -g(\mathbf{x}_n, \beta_k) g(\mathbf{x}_n, \beta_j) \mathbf{x}_n + g(\mathbf{x}_n, \beta_j) \mathbf{x}_n \\ &= g(\mathbf{x}_n, \beta_k) (\delta_{jk} - g(\mathbf{x}_n, \beta_j)) \mathbf{x}_n \end{aligned}$$

where $\delta_{jk} = 1$ if $j = k$, 0 otherwise.

Chuck Anderson

Logistic Regression
Why?
Setup
Derivation

• Now

$$\begin{aligned}\nabla_{\beta_j} l(\beta) &= \sum_{n=1}^N \sum_{k=1}^{K-1} \frac{t_{n,k}}{g(\mathbf{x}_n, \beta_k)} \nabla_{\beta_j} g(\mathbf{x}_n, \beta_k) \\ &= \sum_{n=1}^N \left(\sum_{k=1}^{K-1} t_{n,k} \delta_{jk} - g(\mathbf{x}_n, \beta_j) \sum_{k=1}^{K-1} t_{n,k} \right) \\ &= \sum_{n=1}^N \mathbf{x}_n (t_{n,j} - g(\mathbf{x}_n, \beta_j))\end{aligned}$$

- Now

$$\begin{aligned}
 \nabla_{\beta_j} l(\beta) &= \sum_{n=1}^N \sum_{k=1}^{K-1} \frac{t_{n,k}}{g(\mathbf{x}_n, \beta_k)} \nabla_{\beta_j} g(\mathbf{x}_n, \beta_k) \\
 &= \sum_{n=1}^N \left(\sum_{k=1}^{K-1} t_{n,k} \delta_{jk} - g(\mathbf{x}_n, \beta_j) \sum_{k=1}^{K-1} t_{n,k} \right) \\
 &= \sum_{n=1}^N \mathbf{x}_n (t_{n,j} - g(\mathbf{x}_n, \beta_j))
 \end{aligned}$$

- which results in this update rule for β_j

$$\beta_j \leftarrow \beta_j + \alpha \sum_{n=1}^N (t_{n,j} - g(\mathbf{x}_n, \beta_j)) \mathbf{x}_n$$

- Now

$$\begin{aligned}\nabla_{\beta_j} l(\beta) &= \sum_{n=1}^N \sum_{k=1}^{K-1} \frac{t_{n,k}}{g(\mathbf{x}_n, \beta_k)} \nabla_{\beta_j} g(\mathbf{x}_n, \beta_k) \\ &= \sum_{n=1}^N \left(\sum_{k=1}^{K-1} t_{n,k} \delta_{jk} - g(\mathbf{x}_n, \beta_j) \sum_{k=1}^{K-1} t_{n,k} \right) \\ &= \sum_{n=1}^N \mathbf{x}_n (t_{n,j} - g(\mathbf{x}_n, \beta_j))\end{aligned}$$

- which results in this update rule for β_j

$$\beta_j \leftarrow \beta_j + \alpha \sum_{n=1}^N (t_{n,j} - g(\mathbf{x}_n, \beta_j)) \mathbf{x}_n$$

- How do we do this in R?