CS142: Machine Learning
Lecture 11: Gradient Descent, Multinomial Logistic Regression

Instructor: Erik Sudderth
Brown University Computer Science
October 20, 2015

Figure credits:
PRML: Pattern Recognition & Machine Learning, Bishop 2007
ESL: Elements of Statistical Learning, Hastie & Tibshirani & Friedman 2009
CS142 ML: Lecture 11 Outline

- Gradient optimization for logistic regression
- Newton’s method & reweighted least squares
- Bayesian learning for logistic regression
- Multinomial (multiple category) logistic regression
Generative & discriminative models both learn the likelihood ratio used for Bayesian decision theory and ROC curves:

\[
\frac{p(t = 1 \mid x)}{p(t = 0 \mid x)} = \frac{p(t = 1)p(x \mid t = 1)}{p(t = 0)p(x \mid t = 0)}
\]
Logistic Regression Model

\[ p(t_i \mid x_i, w) = \text{Ber}(t_i \mid \sigma(w^T \phi(x_i))) \]

- Linear discriminant analysis: \( d+1 \) features

\[ \phi(x_i) = [1, x_{i1}, x_{i2}, \ldots, x_{id}] \]

Can always encode optimal decision for equal-covariance Gaussians

- Quadratic discriminant analysis: \( d+1+0.5d(d+1) \) features

\[ \phi(x_i) = [1, x_{i1}, \ldots, x_{id}, x_{i1}^2, x_{i1} x_{i2}, x_{i2}^2, \ldots] \]

Can always encode optimal decision for arbitrary Gaussians

- More generally: \( M \) fixed features \( \phi(x) \)

Flexibility to choose features appropriate for non-Gaussian data, as previously seen for linear regression.

\[ \sigma(z) = \frac{1}{1 + e^{-z}} \]
Learning Logistic Regression Models

Posterior model: \( p(t_n | x_n, w) \)

\( t_n \in \{0, 1\} \)

\( \phi(x_n) \in \mathbb{R}^M \)

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

\( p(t_n | x_n, w) = \text{Ber}(t_n | \sigma(w^T \phi(x_n))) \)

**ML Estimate:** \( \hat{w} = \arg \min_w - \sum_n \log p(t_n | x_n, w) \)

**MAP Estimate:** \( \hat{w} = \arg \min_w - \log p(w) - \sum_n \log p(t_n | x_n, w) \)
Learning Logistic Regression Models

Gradient vectors: 
\[ f : \mathbb{R}^M \rightarrow \mathbb{R} \]
\[ \nabla_w f : \mathbb{R}^M \rightarrow \mathbb{R}^M \]
\[ (\nabla_w f(w))_k = \frac{\partial f(w)}{\partial w_k} \]

Optimization of Smooth Functions:
- **Closed form**: Find zero gradient points. These were linear equations for least squares, but generally are non-linear equations without closed forms.
- **Iterative**: Initialize somewhere, use gradients to take steps towards better (by convention, smaller) values

ML Estimate: 
\[ \hat{w} = \arg \min_w - \sum_n \log p(t_n \mid x_n, w) \]

MAP Estimate: 
\[ \hat{w} = \arg \min_w - \log p(w) - \sum_n \log p(t_n \mid x_n, w) \]
A **globally convergent** algorithm is guaranteed to converge to a **local optimum** from any initialization.

Convergence to a **global optimum** is another issue...
Gradient-Based Optimization Software

MAP Estimate: \( \hat{w} = \arg \min_w - \log p(w) - \sum_n \log p(t_n | x_n, w) \)

You provide a function that, for any \( w \), returns:

- **Objective function value**: Scalar, smaller is better.
- **Gradient vector**: Same dimension as parameter vector.
- Sometimes also the **Hessian** (matrix of second derivatives)

Optimization package does the following:

- **Step sizes**: Guarantees descent to local minimum, tries to move quickly.
- **Descent direction**: Quasi-Newton (conjugate gradient, L-BFGS) methods use sequence of gradients to approximate curvature & converge faster.
- **Convergence**: Stop when changes fall below some tolerance.
- **Robustness**: Care to avoid numerical underflow problems.
Convex Sets
\[ \lambda \theta + (1 - \lambda)\theta' \in \mathcal{S}, \quad \forall \lambda \in [0, 1] \]
\[ \theta, \theta' \in \mathcal{S} \]

Convex Functions
\[ f(\lambda \theta + (1 - \lambda)\theta') \leq \lambda f(\theta) + (1 - \lambda)f(\theta') \]

- Equivalently, the set of points above the curve, or epigraph, is a convex set.
- The negative of a convex function is said to be concave.
Convex Functions

\[ f(\lambda \theta + (1 - \lambda)\theta') \leq \lambda f(\theta) + (1 - \lambda) f(\theta') \]

Properties of Convex Objective Functions:

- From any initialization, gradient descent finds global optimum
  - Ignoring possible numerical precision issues
  - Can be very slow (take huge number of iterations) when initialization is poor and/or gradient method is naïve
- A sum of convex functions remains convex
- Linear and logistic regression have convex negative log-posteriors
Properties of Logistic Functions

\[ \sigma(z) = \frac{1}{1 + e^{-z}} \]

- **Symmetry:**
  \[ \sigma(-z) = 1 - \sigma(z) \]

- **Log-odds ratio:**
  \[ \log \frac{\sigma(z)}{1 - \sigma(z)} = z \]
Logistic Regression: ML Estimation

\[ p(t_n \mid x_n, w) = \text{Ber}(t_n \mid \sigma(w^T \phi(x_n))) \]
\[ \mu_n = \sigma(w^T \phi(x_n)) \]

**Goal:** Minimize negative conditional log-likelihood

\[ f(w) = - \log p(t \mid x, w) = - \sum_{n=1}^{N} t_n \log \mu_n + (1 - t_n) \log(1 - \mu_n) \]

\[ t = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix} \in \{0, 1\}^N \quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_N \end{bmatrix} \in \mathbb{R}^N \quad \Phi = \begin{bmatrix} \phi(x_1)^T \\ \phi(x_2)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix} \in \mathbb{R}^{N \times M} \]
**Logistic Regression: ML Estimation**

\[ p(t_n \mid x_n, w) = \text{Ber}(t_n \mid \sigma(w^T \phi(x_n))) \]

\[ \mu_n = \sigma(w^T \phi(x_n)) \]

**Goal:** Minimize negative conditional log-likelihood

\[ f(w) = - \log p(t \mid x, w) = - \sum_{n=1}^{N} t_n \log \mu_n + (1 - t_n) \log(1 - \mu_n) \]

**Gradient:**

\[ \nabla f(w) = \sum_{n=1}^{N} (\mu_n - t_n) \phi(x_n) = \Phi^T (\mu - t) \]

**Optimum:**

\[ \frac{1}{N} \sum_{n=1}^{N} t_n \phi(x_n) = \frac{1}{N} \sum_{n=1}^{N} \mu_n \phi(x_n) = \frac{1}{N} \sum_{n=1}^{N} \sigma(w^T \phi(x_n)) \phi(x_n) \]

\[ \sigma(z) = \frac{1}{1 + e^{-z}} \]
Gradient optimization for logistic regression
Newton’s method & reweighted least squares
Bayesian learning for logistic regression
Multinomial (multiple category) logistic regression
Newton’s Method

\[ f : \mathbb{R}^M \rightarrow \mathbb{R} \]

**Gradient Vector:**

\[ \nabla_w f : \mathbb{R}^M \rightarrow \mathbb{R}^M \]

\[ (\nabla_w f(w))_k = \frac{\partial f(w)}{\partial w_k} \]

**Hessian Matrix (symmetric):**

\[ \nabla^2_w f : \mathbb{R}^M \rightarrow \mathbb{R}^{M \times M} \]

\[ (\nabla^2_w f(w))_{k\ell} = \frac{\partial^2 f(w)}{\partial w_k \partial w_\ell} \]
Newton’s Method

If objective quadratic & convex, finds minimum in a single iteration
For non-quadratic, finds minimum in fewer iterations than gradient descent
Tradeoff: Each iteration of Newton’s method requires more CPU time

1. Initialize $\theta_0$;
2. for $k = 1, 2, \ldots$ until convergence do
3. Evaluate $g_k = \nabla f(\theta_k)$;
4. Evaluate $H_k = \nabla^2 f(\theta_k)$;
5. Solve $H_k d_k = -g_k$ for $d_k$;
6. Use line search to find stepsize $\eta_k$ along $d_k$;
7. $\theta_{k+1} = \theta_k + \eta_k d_k$;
Weighted Least Squares

Ordinary Least Squares:

\[
E(w) = \frac{1}{2} \sum_{n=1}^{N} (t_n - \phi(x_n)^T w)^2 = \frac{1}{2} \|t - \Phi w\|^2
\]

\[
\hat{w} = (\Phi^T \Phi)^{-1} \Phi^T t
\]

Assume \( N \geq M \) and \( \text{rank}(\Phi) = M \).

Weighted Least Squares:

\[
E(w) = \frac{1}{2} \sum_{n=1}^{N} r_n (t_n - \phi(x_n)^T w)^2 = \frac{1}{2} (t - \Phi w)^T R(t - \Phi w)
\]

\[
R = \text{diag}\{ r_1, r_2, \ldots, r_n \}
\]

\( r_n > 0 \) is importance/confidence/weight for observation \( n \)

\[
\hat{w} = (\Phi^T R \Phi)^{-1} \Phi^T R t
\]
Iteratively Reweighted Least Squares

**Newton’s Method:** For logistic regression model, and using superscripts to index algorithm iteration,

\[
\begin{align*}
\omega^{(k+1)} &= \omega^{(k)} + (\Phi^T R^{(k)} \Phi)^{-1} \Phi^T (t - \mu^{(k)}) \\
\mu_n^{(k)} &= \sigma(\phi(x_n)^T \omega^{(k)})
\end{align*}
\]

**Objective:**

\[
f(w) = -\sum_{n=1}^{N} t_n \log \mu_n + (1 - t_n) \log(1 - \mu_n)
\]

**Gradient:**

\[
\nabla f(w) = \sum_{n=1}^{N} (\mu_n - t_n) \phi(x_n) = \Phi^T (\mu - t)
\]

**Hessian:**

\[
\nabla^2 f(w) = \sum_{n=1}^{N} \mu_n (1 - \mu_n) \phi(x_n) \phi(x_n)^T = \Phi^T R \Phi
\]

\[
R = \text{diag}\{\mu_n (1 - \mu_n)\} \in \mathbb{R}^{N \times N}
\]
Iteratively Reweighted Least Squares

**Newton’s Method:** For logistic regression model, and using superscripts to index algorithm iteration,

\[
\begin{align*}
\vec{w}^{(k+1)} &= \vec{w}^{(k)} + (\Phi^T \text{diag}(\mu^{(k)}) \Phi)^{-1} \Phi^T (\vec{t} - \mu^{(k)}) \\
\mu^{(k)}_n &= \sigma(\phi(x_n)^T \vec{w}^{(k)})
\end{align*}
\]

**Interpretation:**
- At each iteration, find new weights by solving a weighted least squares problem (system of linear equations)
- Data close to decision boundary has highest weight!

**Hessian:**

\[
\nabla^2 f(\vec{w}) = \sum_{n=1}^{N} \mu_n (1 - \mu_n) \phi(x_n) \phi(x_n)^T = \Phi^T R \Phi
\]

\[
R = \text{diag}\{\mu_n (1 - \mu_n)\} \in \mathbb{R}^{N \times N}
\]
Logistic Regression versus Least Squares

Linear Regression:
\[ f(w) = \sum_{n=1}^{N} (t_n - w^T \phi(x_n))^2 \]

Logistic Regression:
\[ f(w) = -\sum_{n=1}^{N} t_n \log \mu_n + (1 - t_n) \log(1 - \mu_n) \]
\[ \mu_n = \sigma(w^T \phi(x_n)) \]
\[ \nabla^2 f(w) = \sum_{n=1}^{N} \mu_n (1 - \mu_n) \phi(x_n) \phi(x_n)^T \]
CS142 ML: Lecture 11 Outline

- Gradient optimization for logistic regression
- Newton’s method & reweighted least squares
- Bayesian learning for logistic regression
- Multinomial (multiple category) logistic regression
Logistic Regression ML: Degeneracies

\[
\log \frac{p(t_n \mid x_n, w)}{1 - p(t_n \mid x_n, w)} = w^T \phi(x_n)
\]

\(w_m > 0\): increasing \(\phi_m(x_n)\) makes \(t_n = 1\) more likely

\(w_m < 0\): increasing \(\phi_m(x_n)\) makes \(t_n = 1\) less likely

\(w_m = 0\): changing \(\phi_m(x_n)\) is independent of \(t_n\)

Pathological estimates that poorly generalize:

- If the training data is \textit{linearly separable} in the feature space, conditional ML takes \(\|w\| \to \infty\) (threshold function)
- If a single feature happens to distinguish the two classes, that feature will be given infinite weight
- If multiple features are constant \((\phi_m(x_n) = c_m)\) across the training data, no unique optimum (numerical problems)
MAP Estimation with Gaussian Priors

\[ p(w) = \mathcal{N}(w \mid 0, \alpha^{-1} I) \]

Objective to Minimize:

\[ f(w) = -\log p(t \mid x, w) = -\sum_{n=1}^{N} \log p(t_n \mid x_n, w) \]

\[ \bar{f}(w) = -\log p(t \mid x, w) - \log p(w) = f(w) + \frac{\alpha}{2} w^T w \quad \text{(ignoring constants)} \]

Gradient:

\[ \nabla_w \bar{f}(w) = \nabla_w f(w) + \alpha w \]

Hessian:

\[ \nabla^2_w \bar{f}(w) = \nabla^2_w f(w) + \alpha I \]

Including a Gaussian prior on weights, or equivalently adding L2 regularization, is a simple modification to the gradient vector & Hessian matrix for any model.
Logistic Regression: Gaussian MAP

\[ p(t_n \mid x_n, w) = \text{Ber}(t_n \mid \sigma(w^T \phi(x_n))) \]

\[ \mu_n = \sigma(w^T \phi(x_n)) \]

\[ p(w) = \mathcal{N}(w \mid 0, \alpha^{-1} I) \]

**Goal:** Minimize negative log-posterior distribution

\[ \bar{f}(w) = \frac{\alpha}{2} w^T w - \sum_{n=1}^{N} t_n \log \mu_n + (1 - t_n) \log(1 - \mu_n) \]

**Gradient:**

\[ \nabla \bar{f}(w) = \alpha w + \sum_{n=1}^{N} (\mu_n - t_n) \phi(x_n) \]

\[ w^{(k+1)} = w^{(k)} - \nabla f(w^{(k)}) = w^{(k)} - \alpha w^{(k)} + \sum_{n=1}^{N} (t_n - \mu_n) \phi(x_n) \]
Logistic Regression Likelihood

**Linearly Separable Data**

$$f(w) = - \sum_{n=1}^{N} t_n \log \mu_n + (1 - t_n) \log(1 - \mu_n)$$

$$\mu_n = \sigma(w^T x_n)$$
Logistic Regression MAP Predictions

\[ p(y=1|x, \text{wMAP}) \]

\[ \bar{f}(w) = \frac{\alpha}{2} w^T w - \sum_{n=1}^{N} t_n \log \mu_n + (1 - t_n) \log(1 - \mu_n) \]

\[ \mu_n = \sigma(w^T x_n) \]
Parameter Estimation versus Prediction

\[ p(t_n, x_n) = p(t_n)p(x_n | t_n) = p(x_n)p(t_n | x_n) \]

**TRAINING DATA:**
\[ t_{\text{train}} = \{t_1, \ldots, t_N\} \]
\[ x_{\text{train}} = \{x_1, \ldots, x_N\} \]

**TEST DATA:**
\[ w(\mathbf{t}_n, \mathbf{x}_n) = p(\mathbf{x}_n)p(\mathbf{t}_n | \mathbf{x}_n) = p(\mathbf{t}_n)p(\mathbf{x}_n | \mathbf{t}_n) \]

- Learning via MAP (or ML) parameter estimation:
\[ \hat{w} = \arg \max_{\mathbf{w}} p(w | t_{\text{train}}, x_{\text{train}}) \quad p(t_{\text{test}} | x_{\text{test}}) \approx p(t_{\text{test}} | x_{\text{test}}, \hat{w}) \]

- But if we correctly apply the rules of conditional probability:
\[ p(t_{\text{test}} | x_{\text{test}}, t_{\text{train}}, x_{\text{train}}) = \int w p(t_{\text{test}} | x_{\text{test}}, w)p(w | t_{\text{train}}, x_{\text{train}}) \, dw \]
Unlike linear regression, exact prediction integral does not have simple closed form:

\[ p(t \mid x) = \int_w p(t \mid x, w)p(w) \, dw \approx \frac{1}{S} \sum_{s=1}^{S} p(t \mid x, w^s) \quad w^s \sim p(w) \]
CS142 ML: Lecture 11 Outline

- Gradient optimization for logistic regression
- Newton’s method & reweighted least squares
- Bayesian learning for logistic regression
- Multinomial (multiple category) logistic regression
Consider any generative model for $K$-category classification:

$$p(t_k = 1 \mid x) = \frac{\mu_k p(x \mid t_k = 1)}{\sum_{\ell=1}^{K} \mu_{\ell} p(x \mid t_{\ell} = 1)}$$

$$= \frac{\exp(f_k(x))}{\sum_{\ell=1}^{K} \exp(f_{\ell}(x))}$$

$$f_k(x) = \log \mu_k + \log p(x \mid t_k = 1)$$

The posterior probability then depends on the 

**multinomial logistic or “soft-max” function**:

$$s_k(z) = \frac{\exp(z_k)}{\sum_{\ell=1}^{K} \exp(z_{\ell})}, \quad k = 1, \ldots, K.$$ 

$z_k \in \mathbb{R}, \quad s_k(z) > 0, \quad \sum_{k=1}^{K} s_k(z) = 1.$

**Exponentiate, then normalize!**
Multinomial Logistic (softmax) Function

\[
s_k(z) = \frac{\exp(z_k)}{\sum_{\ell=1}^{K} \exp(z_\ell)}, \quad k = 1, \ldots, K.
\]

\[z_k \in \mathbb{R}, \quad s_k(z) > 0, \quad \sum_{k=1}^{K} s_k(z) = 1.
\]

\[s_k(z/T), \quad z = (3, 0, 1)
\]
Softmax versus Logistic Function

Assume $K=2$ categories:

\[ s_1(z) = \frac{e^{z_1}}{e^{z_1} + e^{z_2}} = \frac{1}{1 + e^{z_2-z_1}} = \sigma(z_1 - z_2) \]

\[ s_2(z) = \frac{e^{z_2}}{e^{z_1} + e^{z_2}} = 1 - s_1(z) = \sigma(z_2 - z_1) \]

Binary logistic function is a special case.

\[ \sigma(z) = \frac{1}{1 + e^{-z}} \]

\[ s_k(z) = \frac{\exp(z_k)}{\sum_{\ell=1}^{K} \exp(z_{\ell})}, \quad k = 1, \ldots, K. \]

\[ z_k \in \mathbb{R}, \quad s_k(z) > 0, \quad \sum_{k=1}^{K} s_k(z) = 1. \]
Multinomial Logistic Regression

\[ p(t_n \mid x_n, w) = \text{Cat}(t_n \mid \mu_n) \]

\[ \mu_{nk} = \frac{\exp(w_k^T \phi(x_n))}{\sum_{\ell=1}^{K} \exp(w_{\ell}^T \phi(x_n))} \]

\[ \phi(x_n) \in \mathbb{R}^M \]

\[ w_k \in \mathbb{R}^M \]

\[ \mu_{nk} > 0, \sum_{k=1}^{K} \mu_{nk} = 1. \]

- Uses MK weight parameters to relate every feature to every class.
- Optimal decisions for Gaussian likelihoods with quadratic features:
  \[ \phi(x_n) = \begin{bmatrix} 1, x_{n1}, \ldots, x_{nD}, x_{n1}^2, x_{n1}x_{n2}, x_{n2}^2, \ldots \end{bmatrix} \]
- If Gaussian likelihoods have shared covariance, linear features:
  \[ \phi(x_n) = \begin{bmatrix} 1, x_{n1}, \ldots, x_{nD} \end{bmatrix} \]
Multinomial Logistic Regression

Optimal decisions for Gaussian likelihoods with quadratic features:
\[
\phi(x_n) = [1, x_{n1}, \ldots, x_{nD}, x_{n1}^2, x_{n1}x_{n2}, x_{n2}^2, \ldots]
\]

If Gaussian likelihoods have shared covariance, linear features:
\[
\phi(x_n) = [1, x_{n1}, \ldots, x_{nD}]
\]

\[
p(t_n | x_n, w) = \text{Cat}(t_n | \mu_n)
\]

\[
\mu_{nk} = \frac{\exp(w_k^T \phi(x_n))}{\sum_{\ell=1}^K \exp(w_{\ell}^T \phi(x_n))}
\]

\[
\mu_{nk} > 0, \sum_{k=1}^K \mu_{nk} = 1.
\]

Map probabilities to Red, Green, Blue
Non-Quadratic Feature Functions

\[ \begin{align*}
K &= 5 \\
p(t_n \mid x_n, w) &= \text{Cat}(t_n \mid \mu_n) \\
\mu_{nk} &= \frac{\exp(w_k^T \phi(x_n))}{\sum_{\ell=1}^{K} \exp(w_{\ell}^T \phi(x_n))} \\
\mu_{nk} &> 0, \sum_{k=1}^{K} \mu_{nk} = 1.
\end{align*} \]