IFT 6269: Probabilistic Graphical Models **Fall 2018**

Lecture 8 — September 28

Lecturer: Simon Lacoste-Julien *Scribe: Eeshan Gunesh Dhekane and Younes Driouiche*

Disclaimer: These notes have only been lightly proofread.

8.1 Logistic Regression

About Logistic Regression : Let's turn our attention to the binary classification problem! We define it as the problem of learning a map from the set of input data (usually a subset of $\mathbb{R}^d$ for some $d \in \mathbb{N}$) to the set of two labels (usually denoted by $\{0, 1\}$). There are two major approaches to the classification problem : **Generative** and **Discriminative**. The generative approach models the distribution of the input data along with the distribution of the labels given the input data. The discriminative approach, on the other hand, models only the distribution of the labels given the input data. **Logistic Regression** is a discriminative approach to the problem of binary classification. In this approach, we only model and learn the required distributions of $p(Y \mid X)$, where X and Y represent the random vectors corresponding to the input data and the corresponding labels respectively. Despite this simplicity in the modeling of the problem, logistic regression is a robust approach. This is because many other models share the form of $p(Y \mid X)$ with that of this model. Let us begin our discussion of logistic regression with the mathematical formulation.

Formulation : Let X denote the random vector that corresponds to the input data. We assume that $X \in \mathbb{R}^d$ for some $d \in \mathbb{N}$. Let Y denote the random variable corresponding to the labels of input data. For the binary classification problem, we assign values of 0 and 1 to the two labels. Thus, we have $Y \in \{0, 1\}$. Our goal is to model and learn $p(Y \mid X)$, which is the distribution of the labels given the input data. We model the distributions $p_1 = p(Y = 1 \mid X = \mathbf{x})$ and $p_0 = 1 - p_1 = p(Y = 0 \mid X = \mathbf{x})$ as functions of $\mathbf{x}$. The form of the p_1 distributions is chosen as the **Sigmoid Function** of linear transformation of $\mathbf{x}$ (We will see the reasons later). Thus, we have :

$$p(Y = 1 \mid X = \mathbf{x}) = \sigma\left(\mathbf{w}^\top \mathbf{x}\right), \text{ where } \mathbf{w} \in \mathbb{R}^d \text{ is the parameter of the model} \tag{8.1}$$

This form of $p(Y = 1 \mid X = \mathbf{x})$ gives an expression for the target distribution $p(Y \mid X)$:

$$p(Y = y \mid X = \mathbf{x}) = \left(\sigma\left(\mathbf{w}^\top \mathbf{x}\right)\right)^y \cdot \left(1 - \sigma\left(\mathbf{w}^\top \mathbf{x}\right)\right)^{(1-y)} = \text{Bernoulli}\left(\sigma\left(\mathbf{w}^\top \mathbf{x}\right)\right) \tag{8.2}$$

Now, let $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ denote the given dataset with $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \{0,1\}\ \forall\, i \in \{1,\ldots,n\}$. Then, the goal of learning $p\,(Y \mid X)$ becomes the problem of learning $\mathbf{w}$ from the given dataset D, which we will consider through the next sections.

Generative Motivation [Optional] : We stated earlier that logistic regression is a fairly robust approach. We also defined $p\,(Y = 1 \mid X = \mathbf{x})$ as a function of $\mathbf{x}$ in a particular form. In this subsection, we provide a generative motivation that tries to justify these statements and definitions. For this, we make no major assumptions except for the existence of probability density functions $p\,(\mathbf{x} \mid Y = 1)$ and $p\,(\mathbf{x} \mid Y = 0)$ in $\mathbb{R}^d$ (These distributions are called as **Class-Conditional Distributions**). Starting with the class-conditional distributions, our goal is to obtain $p(Y|X)$. We do in the following manner :

$$p(Y=1|X=\mathbf{x}) = \frac{p(Y=1,X=\mathbf{x})}{p\,(X=\mathbf{x})} = \frac{p(Y=1,X=\mathbf{x})}{p(Y=1,X=\mathbf{x})+p(Y=0,X=\mathbf{x})}$$
$$= \frac{1}{1+\frac{p(Y=0,X=\mathbf{x})}{p(Y=1,X=\mathbf{x})}} = \frac{1}{1+e^{-\log\frac{p(Y=1,X=\mathbf{x})}{p(Y=0,X=\mathbf{x})}}} = \frac{1}{1+e^{-f(\mathbf{x})}} \tag{8.3}$$

Thus, from the previous equation, we get :

$$p(Y=1|X=\mathbf{x}) = \frac{1}{1+e^{-f(\mathbf{x})}} = \sigma(f(\mathbf{x})) \text{ where } \sigma(z) = \tfrac{1}{1+e^{-z}} \text{ is the Sigmoid function} \tag{8.4}$$

Here, $f(\mathbf{x})$ is the **Log-Odds Ratio** and is defined as follows :

$$f(x) = \log\frac{p(Y=1,X=\mathbf{x})}{p(Y=0,X=\mathbf{x})} = \log\underbrace{\frac{p(X=\mathbf{x}|Y=1)}{p(X=\mathbf{x}|Y=0)}}_{\text{Class-Conditional Ratio}} + \underbrace{\log\frac{p(Y=1)}{p(Y=0)}}_{\text{Prior Odds Ratio}} \tag{8.5}$$

Now, note that a major proportion of the common distributions used for modeling are special cases of the **Exponential Family** of distributions (We will study this later in the course). The distribution is specified by two functions $h(\mathbf{x}), T(\mathbf{x})$ and is defined as given below :

$$p_{\text{exp-fam}}(\mathbf{x} \mid \overbrace{\boldsymbol{\eta}}^{\text{Cannonical Parameter}}) = \frac{h(\mathbf{x})}{\exp(\underbrace{A(\boldsymbol{\eta})}_{\text{Log-Partition Function}})} \cdot \exp(\overbrace{\boldsymbol{\eta}^\top \underbrace{T(\mathbf{x})}_{\text{Sufficient Statistics}}}^{\text{Linear in } \boldsymbol{\eta}, T(\mathbf{x})}) \tag{8.6}$$

Let $p(Y=1) = \pi$ and $p\,(X = \mathbf{x} \mid Y = y) = p_{\text{exp-fam}}(\mathbf{x} \mid \boldsymbol{\eta}_y)\ \forall\ y \in \{0,1\}$. Then, we can write $f(\mathbf{x})$ from [8.5] in terms of a **Weight Vector** $\mathbf{w}$ and a **Feature Map** $\phi(\mathbf{x})$ as :

$$f(\mathbf{x}) = \mathbf{w}^\top\phi(\mathbf{x}), \text{ with } \mathbf{w} = \begin{pmatrix} \boldsymbol{\eta}_1 - \boldsymbol{\eta}_0 \\ A(\boldsymbol{\eta}_0) - A(\boldsymbol{\eta}_1) + \log\frac{\pi}{1-\pi} \end{pmatrix} \text{ and } \phi(\mathbf{x}) = \begin{pmatrix} T(\mathbf{x}) \\ 1 \end{pmatrix} \tag{8.7}$$

Thus a generative model with class-conditionals in the exponential family yield $p(Y = 1 \mid X = x) = \sigma\left(\mathbf{w}^\top \phi(\mathbf{x})\right)$, which is precisely the logistic regression model (with feature map $\phi(\mathbf{x})$). As a concrete example (exercise to the reader), with $p\left(X = \mathbf{x} \mid Y = y\right) = \mathcal{N}(\mathbf{x} \mid \mu_y, \Sigma_y)$ (the multivariate Gaussians are an exponential family), then we have 1) if $\Sigma_0 = \Sigma_1$, that $\phi(\mathbf{x}) = \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}$ – this is the linear regression model; 2) otherwise, if you use different covariances for the different classes, then $\phi(\mathbf{x}) = \begin{pmatrix} \mathbf{x}\mathbf{x}^\top \\ \mathbf{x} \\ 1 \end{pmatrix}$ (see also assignment 2)! Note that $h(\mathbf{x})$ in (8.6) does not appear in the definition of $f(\mathbf{x})$, even though it does influence the distribution given by the class-conditional. This means that there are many different generative models which gives the **same** $\sigma\left(\mathbf{w}^\top \phi(\mathbf{x})\right)$ model for $p(Y = 1 \mid X = x)$, and thus the logistic regression model is robust to changes in these choices, which is what we meant by saying that logistic regression is a more robust model than the generative model approach.

8.2 Sigmoid Function and Properties

In this section, we provide a (quick) review of some of the properties of the Sigmoid function. It is formally defined as follows :

$$\sigma \;:\;]-\infty, +\infty[\;\rightarrow\;]0, 1[\text{ and } \sigma\left(z\right) = \frac{1}{1 + e^{-z}} \quad \forall\, z \in]-\infty, +\infty[\tag{8.8}$$

The figure [8.1] shows the graph of the Sigmoid function (over $[-10, +10]$).

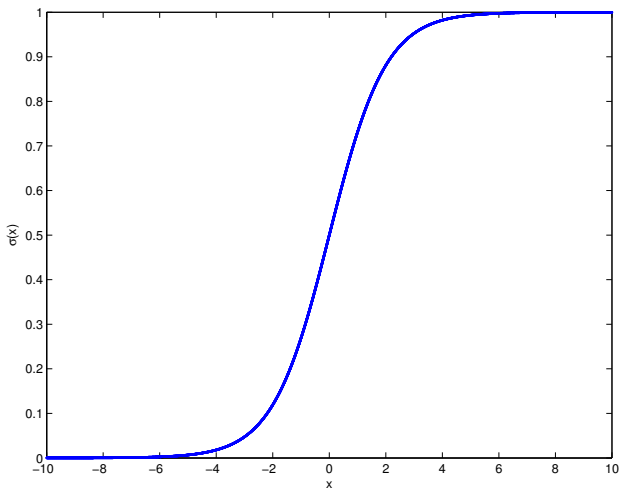

Figure 8.1: Sigmoid function.

Below are some important properties of the Sigmoid function (Prove using [8.8]. Exercise!)

Property 8.2.1 $\forall z \in \mathbb{R}, \sigma(-z) = 1 - \sigma(z)$

Property 8.2.2 $\forall z \in \mathbb{R}, \sigma'(z) = \sigma(z)(1 - \sigma(z)) = \sigma(z)\sigma(-z)$

Property 8.2.3 $\lim_{z \to -\infty} \sigma(z) = 0$ *and* $\lim_{z \to +\infty} \sigma(z) = 1$

8.3 Maximum Conditional Likelihood

Recap : From subsection **Formulation** [8.1], we have the following model for $p(Y \mid X)$:

$$p(y=1|\mathbf{x}) = \sigma\left(\mathbf{w}^\top \mathbf{x}\right), p(y=0|\mathbf{x}) = 1 - \sigma\left(\mathbf{w}^\top \mathbf{x}\right) = \sigma(-\mathbf{w}^\top \mathbf{x}) \text{ and thus,}$$
$$Y|X = \mathbf{x} \sim \text{Bernoulli}\left(\sigma(\mathbf{w}^\top \mathbf{x})\right). \text{ Equivalently, } p(y|\mathbf{x}) = \left(\sigma(\mathbf{w}^\top \mathbf{x})\right)^y \left(\sigma(-\mathbf{w}^\top \mathbf{x})\right)^{1-y} \tag{8.9}$$

Maximum (Conditional) Likelihood : From the subsection **Formulation** [8.1], we also saw that the problem of modeling $p(Y \mid X)$ becomes the problem of learning $\mathbf{w}$ from the dataset D. We will use the method of Maximum Conditional Likelihood to estimate the parameter $\mathbf{w}$ of the model. Given the dataset $D = (\mathbf{x}_i, y_i)_{i=1}^n$, the (conditional) log-likelihood is :

$$\begin{aligned} \ell(\mathbf{w}) &= \sum_{i=1}^n \log(p(y_i|\mathbf{x}_i; \mathbf{w}) \\ &= \sum_{I=1}^n \left[y_i \log\left(\sigma(\mathbf{w}^\top \mathbf{x}_i)\right) + (1 - y_i) \log\left(\sigma(-\mathbf{w}^\top \mathbf{x}_i)\right)\right] \end{aligned} \tag{8.10}$$

Now, in order to solve for $\mathbf{w}$, we first need to find the **Stationary Points** of $\ell(\mathbf{w})$, where we have $\nabla_{\mathbf{w}} \ell(\mathbf{w}) = 0$. Towards this, let us define v_i as follows : $v_i = \mathbf{w}^\top \mathbf{x}_i$. Then, we have $\nabla_{\mathbf{w}} \sigma(\mathbf{w}^\top \mathbf{x}_i) = \nabla_{\mathbf{w}} \sigma(v_i) = \mathbf{x}_i \left[\sigma(v_i)\sigma(-v_i)\right]$. Now, we can evaluate $\nabla_{\mathbf{w}} \ell(\mathbf{w})$ as follows :

$$\nabla_{\mathbf{w}} \ell(\mathbf{w}) = \sum_{i=1}^n \mathbf{x}_i \left[y_i \frac{\sigma(v_i)\sigma(-v_i)}{\sigma(v_i)} - (1 - y_i) \frac{\sigma(v_i)\sigma(-v_i)}{\sigma(-v_i)} \right] = \sum_{i=1}^n \mathbf{x}_i \left[y_i (\underbrace{\sigma(-v_i) + \sigma(v_i)}_{=1}) - \sigma(v_i) \right] \tag{8.11}$$

Thus, we get the required expression for $\nabla_{\mathbf{w}} \ell(\mathbf{w})$:

$$\nabla_{\mathbf{w}} \ell(\mathbf{w}) = \sum_{i=1}^n \mathbf{x}_i \left[y_i - \sigma\left(\mathbf{w}^\top \mathbf{x}_i\right)\right] \tag{8.12}$$

Solving for w : It turns out that $\nabla_{\mathbf{w}}\ell(\mathbf{w}) = 0$ is what is known as a **Transcendental Equation**. Such equations are often hard to solve and do not have closed-form solutions. Thus, we are left with the choice of using **Numerical Methods** to find the maximum conditional likelihood estimate.

The next section provides a description of some useful numerical methods.

Remark 8.3.1 *If we consider $Y = \{-1, 1\}$, then we can encode both cases in one equation as follows:*

$$p(y|\mathbf{x}) = \sigma(y \cdot \mathbf{w}^\top \mathbf{x})$$

Remark 8.3.2 *In contrast to the transcendental equation obtained for the logistic expression approach, we had obtained a **Linear Equation** in the case of least square regression approach. Recall that we obtained $\nabla_{\mathbf{w}}\ell(\mathbf{w}) = \sum_{i=1}^{n} \mathbf{x}_i[y_i - \mathbf{w}^\top \mathbf{x}_i]$ and hence, solving for $\mathbf{w}$ via setting $\nabla_{\mathbf{w}}\ell(\mathbf{w}) = 0$ is essentially solving a linear equation in $\mathbf{w}$!*

8.4 Numerical Optimization

Let us start with a function f defined for some variable $\mathbf{w}$ over a domain $\mathcal{D}$. We want to solve the problem of minimizing $f(\mathbf{w})$ over this domain :

$$\text{Minimize } f(\mathbf{w}) \text{ over } \mathcal{D} \equiv \min_{\mathbf{w}\in\mathcal{D}} f(\mathbf{w}) \tag{8.13}$$

In our case $\mathbf{w} \in \mathbb{R}^d$ and thus, the domain is $\mathcal{D} = \mathbb{R}^d$.

Gradient Descent [1st Order Method]

Motivation: The motivation for this approach is the fact that the gradient of a function points in the direction of the maximum increase in the function. Thus, in order to minimize a function, the natural decision is to follow the direction of the maximum decrease in the function. This can be achieved by traveling in the direction opposite to that of the gradient.

Algorithm : The gradient descent algorithm is described below :

1. **Initialize :** $\mathbf{w}_0$.
2. **Update :** $\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t \nabla f(\mathbf{w}_t)$.
3. **Iterate :** If not converged, go to **Update** step.

Here, γ_t is the size step at iteration t. The γ_t is a hyperparameter and needs to be chosen appropriately. Note that if the γ_t has a very *small* value, then the convergence is very slow. However, if γ_t has a very *large* value, then the algorithm may not converge at all (it may diverge, for instance). Thus, we need to have conditions/heuristics to choose γ_t properly.

Some Step Size Rules and Heuristics :

(a) We can set the step-size to be a constant. This constant is chosen to be equal to $\gamma_t = \frac{1}{L}$ where L is the **Lipschitz Constant** of ∇f. The Lipschitz constant of a vector function ∇f is the smallest number L such that for all $\mathbf{w}, \mathbf{w}'$ in the domain of the function, we have :

$$\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}')\| \leq L \|\mathbf{w} - \mathbf{w}'\|$$

(b) **Decreasing Step-Size Rule** : We can set $\gamma_t = \dfrac{C}{t}$ where C is a constant. The heuristic behind this is that we we want to be able to cover all the domain (achieved by having $\sum_t \gamma_t = \infty$). However, we also want not to deviate far away so that we can converge on the solution (achieved by having $\sum_t \gamma_t^2 < \infty$).

(c) We can choose γ_t by solving : $\min_{\gamma \in \mathbb{R}} f(\mathbf{w}_t + \gamma \boldsymbol{d}_t)$ where $\boldsymbol{d}_t$ is the direction for update. This method is called **Line Search**. However, since this approach is in general costly, we can do approximate search[1].

Newton's Method [2nd Order Method]

Motivation : Here, we approximate the given function in terms of its **Quadratic Approximation**. Now, it is relatively easy to optimize a quadratic function rather than the given function, which might not have desirable convex/concave nature. We use **Taylor Expansion** of the given function to obtain the quadratic approximation as follows :

$$f(\mathbf{w}) = \underbrace{f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^\top (\mathbf{w} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^\top H(\mathbf{w}_t)(\mathbf{w} - \mathbf{w}_t)}_{\text{Quadratic Approximation}} + \underbrace{\mathcal{O}\left(\|\mathbf{w} - \mathbf{w}_t\|\right)^3}_{\text{Taylor's Remainder}}$$

$$= Q_t(\mathbf{w}) + \mathcal{O}\left(\|\mathbf{w} - \mathbf{w}_t\|\right)^3$$

$$\text{where } Q_t(\mathbf{w}) = f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^\top (\mathbf{w} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^\top H(\mathbf{w}_t)(\mathbf{w} - \mathbf{w}_t) \quad (8.14)$$

Here, $H(\mathbf{w}_t)$ is the **Hessian** of the function f and $Q_t(\mathbf{w})$ is a quadratic approximation function for $f(\mathbf{w})$ at $\mathbf{w} = \mathbf{w}_t$. The update formula for $\mathbf{w}_{t+1}$ is obtained by minimizing this quadratic approximation $Q_t(\mathbf{w})$:

$$\nabla_{\mathbf{w}} Q_t(\mathbf{w}) = 0 \Rightarrow \nabla f(\mathbf{w}_t) + H(\mathbf{w}_t)(\mathbf{w} - \mathbf{w}_t) = 0 \Rightarrow \mathbf{w} - \mathbf{w}_t = -H^{-1}(\mathbf{w}_t)\nabla f(\mathbf{w}_t) \quad (8.15)$$

[1] For example, we can have **Armijo Line Search and Conditions**. For more details, please refer to the book Convex Optimization by **Stephen Boyd** and **Lieven Vandenberghe**

Damped Newton's Method: in order to **Stabilize** Newton's method, we incorporate a step size of γ_t. The update step is given as follows :

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t H^{-1}(\mathbf{w}_t)\nabla f(\mathbf{w}_t) \tag{8.16}$$

Algorithm : The algorithm for (Damped) Newton's method is given below :

1. **Initialize :** $\mathbf{w}_0$.
2. **Update :** $\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t H^{-1}(\mathbf{w}_t)\nabla f(\mathbf{w}_t)$.
3. **Iterate :** Until some condition is met ($\|\mathbf{w}_{t+1} - \mathbf{w}_t\| \leq \epsilon$). If not, go to **Update** step.

Advantages and Disadvantages :

- **Convergence :** Newton's method usually gives a much faster convergence (in terms of the number of iterations) compared to the gradient descent method. However, each iteration of Newton's method is more costly than that of the gradient descent method. Specifically, gradient descent update takes $\mathcal{O}(d)$ time and space because we need to manipulate $d-$dimensional vectors and gradients. However, Newton's method involves calculation of inverse of Hessian, which requires $\mathcal{O}(d^2)$ space and takes $\mathcal{O}(d^3)$ time. Thus, there is a trade-off in number of iterations till convergence versus the complexity of each iteration.
- **Affine Invariance and Role of H^{-1}:** Newton's method is **Affine Invariant**, which means that it is invariant to the re-scaling of variables. The reason behind this is that the update term of Newton's method has the inverse of Hessian, which transforms the space to make it "well-conditioned". We demonstrate the intuitive benefits of this property and the effect of the presence of Hessian-inverse in the following (optional) subsection.

The Role of H^{-1} [Optional] : We consider a very simple example, where we have $\mathbf{w} \in \mathcal{D} = \mathbb{R}^2$. Let the function to minimize be given by $f(\mathbf{w}) = w_1^2 + w_2^2$, where w_i ($i \in \{1, 2\}$) denotes the $i-$th component of $\mathbf{w}$. Let us try to minimize this function using gradient descent and Newton's method. We compute the gradient and the Hessian for $f(\mathbf{w})$:

$$\nabla f(\mathbf{w}) = 2\begin{bmatrix}(w_t)_1\\(w_t)_2\end{bmatrix} \in \mathbb{R}^2 \text{ and } H(f(\mathbf{w})) = 2\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}, \text{ i.e., } H^{-1}(f(\mathbf{w})) = \frac{1}{2}\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$$
$$\text{Gradient Descent : } \mathbf{w}_{t+1} = \mathbf{w}_t - 2\gamma_t\mathbf{w}_t \Rightarrow \mathbf{w}_{t+1} - \mathbf{w}_t = \Delta\mathbf{w}_t = -2\gamma_t\mathbf{w}_t$$
$$\text{Newton's Method : } \mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t\mathbf{w}_t \Rightarrow \mathbf{w}_{t+1} - \mathbf{w}_t = \Delta\mathbf{w}_t = -\gamma_t\mathbf{w}_t \tag{8.17}$$

Since the Hessian-inverse is proportional to identity matrix, gradient descent and Newton's method have *essentially* the same update steps (except for constants). Further, the update terms $\Delta\mathbf{w}_t$ are proportional to $\mathbf{w}$ and thus, the updates from $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$ are both directly along the direction from $\mathbf{w}_t$ to the global minimum at $(0, 0)$. However, let us re-parameterize $\mathbf{w}$ using $\mathbf{u}$ as $w_1 = \frac{u_1}{a}, w_2 = \frac{u_2}{b}$ $(a \neq b)$. Thus, the same function $f(\mathbf{w})$ is now a function $g(\mathbf{u})$ with $g(\mathbf{u}) = \frac{u_1^2}{a^2} + \frac{u_2^2}{b^2}$. Now, we compute the gradient and Hessian for $g(\mathbf{u})$ as follows :

$$\nabla g(\mathbf{u}) = 2\begin{bmatrix}\frac{(u_t)_1}{a^2}\\\frac{(u_t)_2}{b^2}\end{bmatrix} \in \mathbb{R}^2 \text{ and } H(g(\mathbf{u})) = 2\begin{bmatrix}\frac{1}{a^2} & 0\\0 & \frac{1}{b^2}\end{bmatrix}, \text{ i.e., } H^{-1}(g(\mathbf{u})) = \frac{1}{2}\begin{bmatrix}a^2 & 0\\0 & b^2\end{bmatrix}$$
$$\text{Gradient Descent : } \mathbf{u}_{t+1} = \mathbf{u}_t - 2\gamma_t\begin{bmatrix}\frac{(u_t)_1}{a^2}\\\frac{(u_t)_2}{b^2}\end{bmatrix} \Rightarrow \mathbf{u}_{t+1} - \mathbf{u}_t = \Delta\mathbf{u}_t = -2\gamma_t\begin{bmatrix}\frac{(u_t)_1}{a^2}\\\frac{(u_t)_2}{b^2}\end{bmatrix}$$
$$\text{Newton's Method : } \mathbf{u}_{t+1} = \mathbf{u}_t - \gamma_t\begin{bmatrix}a^2 & 0\\0 & b^2\end{bmatrix}\begin{bmatrix}\frac{(u_t)_1}{a^2}\\\frac{(u_t)_2}{b^2}\end{bmatrix} \Rightarrow \mathbf{u}_{t+1} - \mathbf{u}_t = \Delta\mathbf{u}_t = \begin{bmatrix}u_1\\u_2\end{bmatrix} \tag{8.18}$$

Now, we see that $\Delta\mathbf{u}_t$ for gradient descent is not proportional to $\mathbf{u}_t$ because $a \neq b$. Hence, the direction of the update $\Delta\mathbf{u}_t$ is not *ideal* because it does not point towards the global minimum at $(0, 0)$. This is the effect of re-parameterization. However, note that the presence of $H^{-1}(g(\mathbf{u}))$ makes the update term of Newton's method proportional to $\mathbf{u}_t$. This makes the direction of the update once again point towards the global minimum. Thus, we can *see* that Newton's method is affine invariant due to presence of the Hessian-inverse in the update step. The figures [8.2] and [8.3] illustrate the comparison of gradient descent updates and Newton method updates for an elliptic loss function and a circular loss function respectively.

8.5 IRLS : Iterative Reweighted Least Square

Formulation : Newton's method applied to logistic regression is often called as IRLS. We use Newton's method to solve the transcendental equation we encountered earlier. Recall

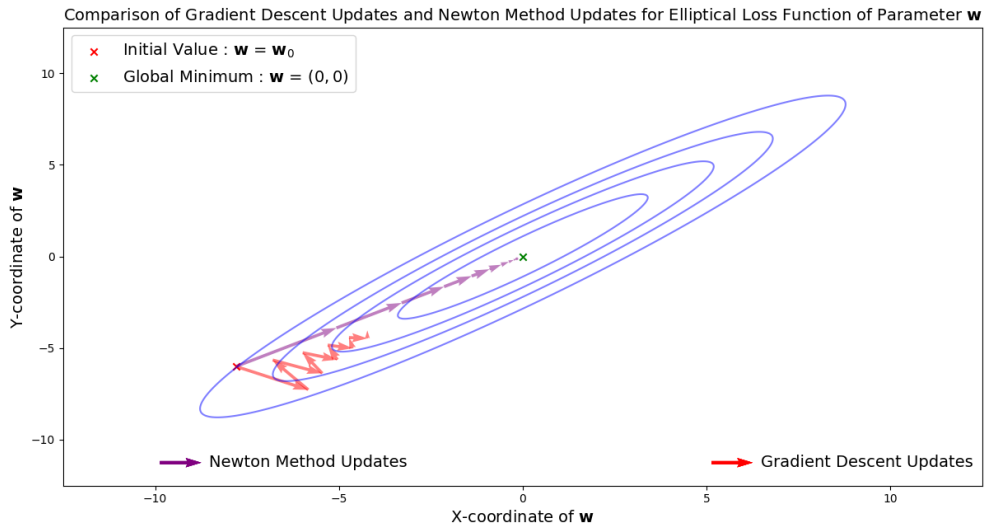

Figure 8.2: For an elliptic loss function, Newton method updates point in the ideal direction, whereas the gradient descent updates do not. This demonstrates the role played by the inverse of the Hessian in Newton method updates.

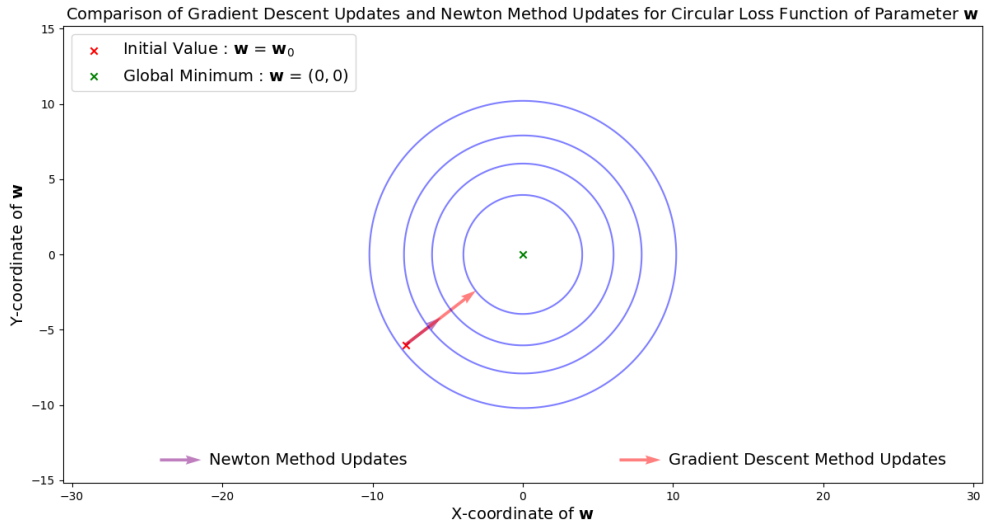

Figure 8.3: The gradient descent updates and Newton method updates are *essentially* identical (except for a scaling constant). For a given initialization of the parameter, they both point in the ideal direction and are coincident.

from [8.12] that $\nabla\ell(\mathbf{w}) = \sum_{i=1}^{n} \mathbf{x}_i[y_i - \sigma(\mathbf{w}^\top x_i)]$. Let $v \in \mathbb{R}^d$ be any given vector. Then :

$$H = H(\ell(\mathbf{w})) = -\sum_{1\le i\le n} \mathbf{x}_i\mathbf{x}_i^\top \sigma(\mathbf{w}^\top\mathbf{x}_i)\sigma(-\mathbf{w}^\top\mathbf{x}_i)$$

$$\Rightarrow v^\top Hv = -\sum_{1\le i\le n} (v^\top\mathbf{x}_i)(\mathbf{x}_i^\top v)\sigma(\mathbf{w}^\top\mathbf{x}_i)\sigma(-\mathbf{w}^\top\mathbf{x}_i) = -\sum_{1\le i\le n} \underbrace{(v^\top\mathbf{x}_i)^2}_{\ge 0}\underbrace{\sigma(\mathbf{w}^\top\mathbf{x}_i)}_{>0}\underbrace{\sigma(-\mathbf{w}^\top\mathbf{x}_i)}_{>0}$$

$$\Rightarrow v^\top Hv \le 0 \;\forall\; v \in \mathbb{R}^d \Rightarrow H\left(\ell(\mathbf{w})\right) \text{ is negative semi-definite}$$

$$\Rightarrow \ell(w) \text{ is } \textbf{concave} \Rightarrow \text{ Newton's updates would indeed maximize log-likelihood.} \tag{8.19}$$

Let $\mathbf{X} = \begin{bmatrix} \vdots \\ —\mathbf{x}_i^\top— \\ \vdots \end{bmatrix}_{n\times d}$ be the **Design Matrix**, $\mathbf{y} = \begin{bmatrix} \vdots \\ y_i \\ \vdots \end{bmatrix}_{d\times 1}$ and $\boldsymbol{\mu} = \begin{bmatrix} \vdots \\ \mu_i \\ \vdots \end{bmatrix}_{d\times 1}$,
where $\forall\; i \in \{1,\ldots,n\}$, we have $\mu_i = \sigma(\mathbf{w}^\top\mathbf{x}_i)$, $\mathbf{x}_i$ is the $i-$th input data and y_i is the label corresponding to $\mathbf{x}_i$. Then, $\nabla\ell(\mathbf{w})$ and $H(\ell(\mathbf{w}))$ can be expressed as :

$$\nabla\ell(\mathbf{w}) = \sum_{i=1}^{n} \mathbf{x}_i(y_i - \mu_i) = \mathbf{X}^\top(\mathbf{y} - \boldsymbol{\mu}) \quad \text{and} \quad H(\ell(\mathbf{w})) = -\mathbf{X}^\top D(\mathbf{w})\mathbf{X} \tag{8.20}$$

Here, D is the $n \times n$ diagonal matrix defined as : $D_{ii} = \mu_i(1-\mu_i)\;\forall\; i \in \{1,\ldots,d\}$. Based on this notation, the Newton's method updates are given as follows :

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - (-\mathbf{X}^\top D_t\mathbf{X})^{-1}\mathbf{X}^\top(\mathbf{y} - \boldsymbol{\mu}_t) \\ &= (\mathbf{X}^\top D_t\mathbf{X})^{-1}\left[(\mathbf{X}^\top D_t\mathbf{X})\mathbf{w}_t + \mathbf{X}^\top(\mathbf{y} - \boldsymbol{\mu}_t)\right] = (\mathbf{X}^\top D_t\mathbf{X})^{-1}(\mathbf{X}^\top D_t\mathbf{z}_t)\end{aligned} \tag{8.21}$$

Here, $\mathbf{z}_t$ is defined as follows : $\mathbf{z}_t = \mathbf{X}\mathbf{w}_t + D_t^{-1}(\mathbf{y} - \boldsymbol{\mu}_t)$. This definition of $\mathbf{z}_t$ along with the update step expression from [8.21] indicate that at each time-step t, we are essentially solving a **(Weighted) Least Square Problem**. This can be seen as follows :

Let $D_t^{\frac{1}{2}}$ be the square matrix (matrix of square root of diagonal entries) of D_t

Since D_t is diagonal, so is $D_t^{\frac{1}{2}}$. Thus, we have $D_t = D_t^\top, D_t^{\frac{1}{2}} = \left(D_t^{\frac{1}{2}}\right)^\top$ (8.22)

Let $\hat{\mathbf{X}}$ be the new design matrix defined as : $\hat{\mathbf{X}} = D_t^{\frac{1}{2}}\mathbf{X}$ (8.23)

Let $\hat{\mathbf{y}}$ be the new design matrix defined as : $\hat{\mathbf{y}} = D_t^{\frac{1}{2}}\mathbf{z}_t$ (8.24)

Then, $\mathbf{w}_{t+1} = \left(\mathbf{X}^\top D_t \mathbf{X}\right)^{-1}\left(\mathbf{X}^\top D_t \mathbf{z}_t\right) \iff \mathbf{w}_{t+1} = \left(\hat{\mathbf{X}}^\top \hat{\mathbf{X}}\right)^{-1}\hat{\mathbf{X}}^\top \hat{\mathbf{y}}$

From previous equation and the **Least Squares Estimation** from previous class :

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \left\|\hat{\mathbf{y}} - \hat{\mathbf{X}}\mathbf{w}\right\|^2 = \arg\min_{\mathbf{w}} \left\|D^{\frac{1}{2}}(\mathbf{z}_t - \mathbf{X}\mathbf{w})\right\|^2 = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \frac{\left((\mathbf{z}_t)_i - \mathbf{x}_i^\top \mathbf{w}\right)^2}{D_{ii}^{-1}}$$

Thus, $\mathbf{w}_{t+1}$ is the solution of the above weighted least squares problem. (8.25)

$\frac{\left((\mathbf{z}_t)_i - \mathbf{x}_i^\top \mathbf{w}\right)^2}{D_{ii}^{-1}}$ has the form of $\frac{\left(y_i - \boldsymbol{x_i}^\top \boldsymbol{w}\right)^2}{\sigma^2}$, which represents the **Gaussian Noise Model**

Thus, D_{ii}^{-1} takes the form of **Data-Dependent Noise**. (8.26)

8.6 Logistic Regression for Big Data

Big Data and Constraints on Logistic Regression : The term **Big Data** often stands for datasets with a large number of data points (large value of n), each element being a vector in a high-dimensional space (large value of d). With big data, there are several constraints on the methods used to model the dataset using logistic regression.

- We have seen that the second order Newton's method incurs $\mathcal{O}(d^3)$ time and $\mathcal{O}(d^2)$ space due to the computation of the inverse of the Hessian. In the case of big data, d is a large number and thus, we cannot afford these order of computation. Thus, we must resort to the first order methods.
- Now, we have also seen that the first order method of gradient descent has faster iterations. However, the number of iterations till convergence is large. Thus, we need to improve upon the gradient descent method.
- **Batch Gradient Descent :** We consider the so-called **Batch Gradient Descent**, the update step of which is described as follows–

$$\textbf{Update : } \mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t \left(\frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\mathbf{w}_t) \right), \quad \text{where } f_i(\mathbf{w}_t) \text{ represents the gradient at the } i-\text{the input feature} \tag{8.27}$$

However, the computation of each iteration involves $\mathcal{O}(d)$ computations for gradients of n features. Thus, the overall computations per iteration are $\mathcal{O}(n \cdot d)$, which can not be afforded. Thus, we can not use batch gradient descent for big data.

Thus, we resort to the use of the **Stochastic Gradient Descent** and its variants/extensions. We discuss some of the representative methods in the following subsections.

Stochastic Gradient Descent [SGD] : The update step of SGD involves randomly picking up an input feature i_t at iteration t and plugging in the gradient descent formula the gradient for that feature. Formally, the update formula for SGD is given by :

$$\begin{aligned} \textbf{Update : } & \text{Randomly pick } i_t \text{ at iteration/time-instance } t \\ & \text{Evaluate the gradient } \nabla f_{i_t}(\mathbf{w}_t) \text{ at the input feature indexed } i_t \\ & \text{Update } \mathbf{w} \text{ as follows : } \quad \mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \nabla f_{i_t}(\mathbf{w}_t) \end{aligned} \tag{8.28}$$

Now, this approach has cheaper complexity; $\mathcal{O}(d)$ computations are performed at each iteration. However, the convergence of this method is very poor when compared against the batch gradient descent. The method also has a very high variance. This can be intuitively seen as follows. If there are certain inputs for which the gradient takes abnormally large values, then the update step will sway $\mathbf{w}$ from its *ideal* update direction by huge amounts. This is not good for the convergence, since such samples will easily mislead the updates. To some extent, we can try to cater for the high variance by using **Stochastic Mini-Batch Gradient Descent**. Here, instead of evaluating gradient at a single randomly picked feature, we evaluate averaged gradient over a randomly selected batch of features with indices $\mathcal{B} = \{i_1, \ldots, i_b\}$. The update step is described below :

$$\begin{aligned} \textbf{Update : } & \text{Randomly pick indices } \mathcal{B} = \{i_1, \ldots, i_b\} \text{ at iteration } t && (8.29) \\ & \text{Evaluate the averaged gradient } g = \tfrac{1}{b} \textstyle\sum_{j=1}^{b} \nabla f_{i_j}(\mathbf{w}_t) && (8.30) \\ & \text{Update } \mathbf{w} \text{ as follows : } \quad \mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \cdot g && (8.31) \end{aligned}$$

The updates now incur $\mathcal{O}(b \cdot d)$ computations per iteration and for appropriate choice of b, we can afford these updates. Intuitively, averaging out of the gradients diminishes the effects of abnormally large gradients on the update of $\mathbf{w}$, which slightly improves (decreases) the variance. Note that the convergence analysis of mini-batch SGD shows that there is no advantage to use a mini-batch size bigger than 1 *if the cost of a mini-batch step is b times the cost for one gradient*. The reason is that the variance decreases when b increases, yielding a smaller total number iterations to reach a specific accuracy. However, this reduction of number iterations is smaller than b and thus, if each step is b times more expansive, there is no overall gain of using a mini-batch. The *main computational reason* to use mini-batch is when one has access to **Parallel Processing**. With parallel processors, the computation of the b gradients can be made almost as fast as that of only one gradient and this yields an overall speed-up. Note that SGD is an example of a set of approaches called **Incremental Gradient Methods** in optimization.

In the last 5 years, a set of methods called **Variance Reduced Incremental Gradient Methods** were proposed to further speed-up these methods. Their idea is to reduce the variance by using "memory". We now present the first one of these (which was a big breakthrough in the optimization literature[2]), Stochastic Averaged Gradient (SAG), and a small tweak called SAGA:

Stochastic Averaged Gradient [SAG] and SAGA : The idea of SAG is to use the memory of the previous computations to calculate the update term. The update is :

$$\textbf{Update : } \mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \frac{1}{n} \sum_{i=1}^{n} v_i \tag{8.32}$$

Here, $v_i = \nabla f_i(\mathbf{w}_t)$ are called **Memory** variables. In every iteration, we update only one of the memory variables $v_{i_t} = \nabla f_{i_t}(\mathbf{w}_t)$ for some randomly picked i_t. We keep all the previously set v_j memory variables unchanged and simply use their computed values from previous iterations. This method of incrementally computing the approximated gradient decreases the variance of the method considerably, and in particular, allows the use of a big *constant step-size* like gradient descent which is why convergence is so much better than SGD. A disadvantage of SAG was that in expectation over the random choice i_t, the update direction is not equal to $\nabla f(\mathbf{w}_t)$ (it is **Biased**). Because of that, the convergence proof was very complicated with tens of pages with numerically found quantities.

The **SAGA** method is a small change on SAG to make the update direction **Unbiased**, significantly simplifying its convergence proofs (a few lines!). The update step for SAGA is:

[2]The paper **Minimizing Finite Sums with the Stochastic Average Gradient** won the Lagrange prize in mathematical programming.

$$\textbf{Update : } \mathbf{w}_{t+1} = \mathbf{w}_t - \gamma(\nabla f_{i_t}(\mathbf{w}_t) + \underbrace{\frac{1}{n}\sum_{i=1}^{n} v_i - v_{i_t}}_{\substack{\textbf{Variance Reducing} \\ \textbf{Correction}}}) \tag{8.33}$$

One can see that the variance reducing correction term is zero in expectation over i_t, thus yielding an unbiased update direction $\nabla f(\mathbf{w}_t)$ in expectation. SAGA is now the default method for optimizing logistic regression in **Scikit-learn** library[3]. The figure [8.4] summarizes the characteristics of the methods discussed above.

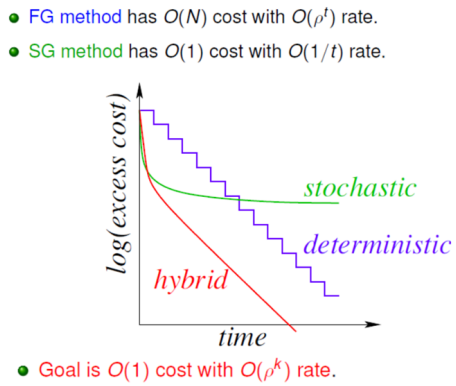

Figure 8.4: A comparison of the Batch Gradient (FG method, deterministic), Stochastic Gradient method (SG method, stochastic) and SAG/SAGA (hybrid) methods. The FG method converges faster, but each update is costly. The SG method has cheap updates, but it converges slowly. The hybrid methods are the best of both worlds; these methods achieve a faster convergence with cheap updates.

To go further : The above notes were just a quick introduction on the topic. These methods are covered in greater details in the class **IFT6132 - Advanced Structured Prediction and Optimization**.

[3]Scikit-learn is one of the standard libraries for python-based machine learning. For more information on SAGA, please refer to the original **NIPS 2014 Paper**: **SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives**.