CS 6140: Machine Learning **Spring 2015**
College of Computer and Information Science
Northeastern University
Lecture 2 **January, 26**
Instructor: Bilal Ahmed **Scribe: Bilal Ahmed & Yu Wen**

Logistic Regression[1]

1 Introduction

An example: Suppose we have a classification problem: a bank wants to predict whether a credit card transaction is a fraud or not. The bank has some related information such as the transaction price, the mean price of recent transactions, the number of recent transactions and etc. Given these features, how can we decide whether this transaction is fraudulent or not? As opposed to the linear regression problem that we studied in the previous modules, here we a discrete output variable $y \in 0, 1$, where $y = 1$ indicates that transaction is fraudulent. When the output variable is discrete we will use the framework of classification instead of regression.

We can convert out problem of predicting a discrete value, to a continuous one, by predicting the probability that y is one, i.e., $P(y = 1|x)$. Subsequently, we can threshold the predicted probability estimate to make the final binary prediction. Similar to linear regression, we can use the linear combination of the given input features to create a linear classification algorithm. Once we have calculated $w^T x$, we can use a function to map the linear combination to lie between 0 and 1. This final score we will interpret as $P(y = 1|x, w)$, where w is the weight vector defined in a manner similar to linear regression.

Functions that have the property of mapping arbitrary input values to a limited range, $[0, 1]$ in our case are known as squashing functions. In this module we will focus on the sigmoid function as our choice of squashing function.

2 The Sigmoid Function

The sigmoid function takes an input which ranges from negative infinity to positive infinity and output a number between 0 and 1, as followings.
More specifically, the sigmoid function is defined as

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

Since we will develop a gradient descent algorihtm for logistic regression, it is convenient to look at the derivative of the sigmoid function, which we can use later.

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{2}$$

[1]Based on lecture notes by Andrew Ng. These lecture notes are intended for in-class use only.

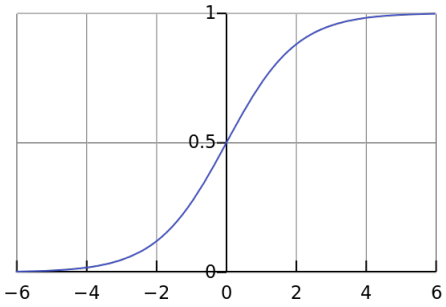

Figure 1: the sigmoid function

We can derive this as follows:

$$\begin{aligned}
\sigma'(x) &= \frac{d}{dx}\frac{1}{1+e^{-x}} \\
&= \frac{1}{(1+e^{-x})^2}e^{-x} \\
&= \frac{1}{1+e^{-x}}\frac{1+e^{-x}-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}}(1-\frac{1}{1+e^{-x}}) \\
&= \sigma(x)(1-\sigma(x))
\end{aligned}$$

3 Piecing It Togetjer:

Let the design matrix (training data) be formulated as:

$$\mathbf{X} = \begin{bmatrix} x_{10} & x_{11} & x_{12} & \dots & x_{1m} \\ x_{20} & x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N0} & x_{N1} & x_{N2} & \dots & x_{Nm} \end{bmatrix}$$

,where the i^{th} row $\mathbf{x}_i \in \mathbb{R}^{m+1}$, represents the i^{th} example. And the frist column of $\mathbf{X}$ is the dummy feature such that $\mathbf{x}_{i0} = 1$ for $i = 1...N$.

Similarly, let w be the weight vector which represents the feature coefficients (weights), defined as:

$$\mathbf{w} = (w_0, w_1, w_2, ..., w_m)$$

corresponding to the $m+1$ input features. Note that, w_0 corresponds to the bias term (intercept). Furthermore, we will arrange the labels of the N training examples arranged as a vector $\mathbf{y} \in \mathbb{R}^N$:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Then, the logistic regression model can be described as:

$$P(y_i = 1|\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x}_i) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}_i}} \tag{3}$$

where $P(y_i = 1|\mathbf{x}_i, \mathbf{w})$ is the probability of $y_i = 1$ given the i^{th} input example $\mathbf{x}_i$, and model parameters $\mathbf{w}$.

Recall, that the output of the sigmoid function is the probability of y_i being equal to 1. But if we are going to make a hard decision such as $y_i = 1$, we need to threshold the output probability. For example, if we set the threshold to be 0.5 such that if $P(y_i = 1|\mathbf{x}_i)$ is greater than 0.5 we say $y_i = 1$.

4 Estimating Model Parameters

Given input training data, how can we learn/train/fit our model to data? Another way to pose this question is how do we estimate the optimal values of $\mathbf{w}$ given the training data? We will use the maximum likelihood framework for estimating model parameters (we have used max. likelihood previously to estimate the model parameters for linear regression).

4.1 Maximum Likelihood Parameter Estimation

Since, $P(y_i = 1|\mathbf{x}_i, w) = \sigma(\mathbf{w}^T\mathbf{x}_i)$, and $y_i \in 0, 1$ we have:

$$P(y_i = 0|\mathbf{x}_i, w) = 1 - P(y_i = 1|\mathbf{x}_i, w) = 1 - \sigma(\mathbf{w}^T\mathbf{x}_i) \tag{4}$$

If we combine equation 3 and 4 together, we get:

$$P(y_i|\mathbf{x}_i, w) = \sigma(\mathbf{w}^T\mathbf{x}_i)^{y_i}(1 - \sigma(\mathbf{w}^T\mathbf{x}_i))^{1-y_i} \tag{5}$$

Convince yourself that the above class probability estimate agrees with our model, by using the fact that y_i can either be 0 or 1.
We will further assume that our training data consists of data that has been independently sampled from the same underlying distribution (data is I.I.D.). This assumption allows us to write the likelihood of the training data as:

$$L(\mathbf{w}) = \prod_{i=1}^{N} \sigma(\mathbf{w}^T\mathbf{x}_i)^{y_i}(1 - \sigma(\mathbf{w}^T\mathbf{x}_i))^{1-y_i} \tag{6}$$

where $\mathbf{x}_i$ and y_i is the input vector and the label of the i^{th} example, respectively.

The optimal weight vector $\mathbf{w}^*$ is obtained by maximizing the data likelihood:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \prod_{i=1}^{N} \sigma(\mathbf{w}^T\mathbf{x}_i)^{y_i}(1 - \sigma(\mathbf{w}^T\mathbf{x}_i))^{1-y_i} \tag{7}$$

Instead of maximizing the likelihood directly we can equivalentlty minimize the negative log-likelihood:

$$E(\mathbf{w}) = \sum_{i=1}^{N} -y_i \ln(\sigma(\mathbf{w}^T\mathbf{x}_i)) - (1-y_i)\ln(1-\sigma(\mathbf{w}^T\mathbf{x}_i)) \tag{8}$$

the optimal weight thus obtained is

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w}) \tag{9}$$

4.2 Optimizing the Objective Function

As a first approach, we will use the gradient descent algorithm to minimize the negative log-likelihood (Equation 8).

4.2.1 Gradient Descent

Recall that the gradient descent update rule is

$$\mathbf{w}^{new} = \mathbf{w}^{old} - \alpha \nabla_{\mathbf{w}} E(\mathbf{w}^{old})$$

,where α is the learning rate. $\nabla_{\mathbf{w}} E(\mathbf{w})$ is the gradient of $E(\mathbf{w})$, which is a vector.
A The j^{th} component of $\nabla_{\mathbf{w}} E(\mathbf{w})$, which is defined as $\frac{\delta E(\mathbf{w})}{\delta w_j}$, is the partial derivative of $E(\mathbf{w})$ with respect to w_j (the j^{th} component of $\mathbf{w}$)

Thus, for the w_j the update rule is

$$w_j = w_j - \alpha \frac{\delta E(\mathbf{w})}{\delta w_j} \qquad 0 \leq j \leq m$$

where,

$$\frac{\delta E(\mathbf{w})}{\delta w_j} = \sum_{i=1}^{N} (\sigma(\mathbf{w}^T\mathbf{x}_i) - y_i) x_{ij} \tag{10}$$

Thus, the gradient descent update rule for w_j is

$$w_j = w_j - \alpha \sum_{i=1}^{N} (\sigma(\mathbf{w}^T\mathbf{x}_i) - y_i) x_{ij} \qquad j = 0...m \tag{11}$$

We can vectorize the above update procedure to perform a simultaneous update across all input features. Let $\begin{bmatrix} \sigma(\mathbf{w}^T\mathbf{x}_1) \\ \sigma(\mathbf{w}^T\mathbf{x}_2) \\ \vdots \\ \sigma(\mathbf{w}^T\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_N \end{bmatrix} = \mathbf{o}$ The gradient (Equation 10), can be written as:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \mathbf{X}^T(\mathbf{o} - \mathbf{y}) \tag{12}$$

Thus, the gradient descent update rule becomes:

$$\mathbf{w}^{new} = \mathbf{w}^{old} - \alpha \mathbf{X}^T(\mathbf{o} - \mathbf{y}) \tag{13}$$

4.2.2 Newton-Raphson Method

Recall that we are interested in finding the optimal model paramters $\mathbf{w}^*$ by minimizing $E(\mathbf{w})$. At w^*, the gradient of $E(\mathbf{w})$ which is $\nabla_{\mathbf{w}} E(\mathbf{w})$ is a zero vector. Then, our optimization goal can be equivalently viewed as finding $\mathbf{w}^*$ such that $\nabla_{\mathbf{w}} E(\mathbf{w}^*) = 0$. Note that, here "0" means the zero vector instead of a zero real value.
Newton-Raphson method, also known as **Newton's method**, is an iterative method for finding roots of a function(finding a x^* such that $f(x^*) = 0$). If we view the gradient of $E(\mathbf{w})$ as a function, we can use the Newton-Raphson method to find the $\mathbf{w}^*$ such that $\nabla_{\mathbf{w}} E(\mathbf{w}^*) = 0$.

Newton-Raphson Method: Newton-Raphson method is an iterative method. It starts at some initial guess of x^* .Then, at each step, it update its current estimate of x^* by using the follwing update rule,

$$x^{new} = x^{old} - \frac{f(x^{old})}{f'(x^{old})}$$

, where $f'(x)$ is the derivative of $f(x)$. The process will converge to some x^{new} such that $f(x^{new}) = 0$.

For example, suppose we want to find the root of the function

$$f(x) = x^2 - 4$$

We start at some random point, for example, $x = 5$ and iteratively apply above rule. Then we will converge at the point $x = 2$ such that $f(2) = 0$. Details are as followings

$iteration$	x^{old}	$x^{new} = x^{old} - \frac{(x^{old})^2 - 4}{2x^{old}}$
1	5	2.9
2	2.9	2.139655172413793
3	2.139655172413793	2.0045576426130207
4	2.0045576426130207	2.0000051812194735
5	2.0000051812194735	2.000000000006711
6	2.000000000006711	2

As we discussed earlier, we will use Newton's method to find $\mathbf{w}^*$ such that $\nabla_{\mathbf{w}} E(\mathbf{w}^*) = 0$. We need to find the derivative of $\nabla_{\mathbf{w}} E(\mathbf{w})$, which will be second order derivative of $E(\mathbf{w})$. Then, the update rule for the Newton-Raphson method will be:

$$\mathbf{w}^{new} = \mathbf{w}^{old} - \frac{\nabla_{\mathbf{w}} E(\mathbf{w}^{old})}{\nabla_{\mathbf{w}}(\nabla_{\mathbf{w}} E(\mathbf{w}))} \tag{14}$$

,where $\nabla_{\mathbf{w}}(\nabla_{\mathbf{w}}E(\mathbf{w}))$ is a matrix that represents the second order derivatives of $E(\mathbf{w})$ which is known as the Hessian matrix.

The Hessian Matrix: The derivative of $\nabla_{\mathbf{w}}E(\mathbf{w})$ is called the **Hessian matrix**. Since $\nabla_{\mathbf{w}}E(\mathbf{w})$ is a vector and $\nabla_{\mathbf{w}}E(\mathbf{w}) \in \mathbb{R}^{m+1}$, the derivatives of this vector w.r.t $\mathbf{w}$ is an $(m+1)\times(m+1)$ matrix, whose ij^{th} entry is:

$$H_{ij} = \frac{\delta E(\mathbf{w})}{\delta w_i w_j}$$

More specifically, the Hessian matrix $H \in \mathbb{R}^{(m+1)\times(m+1)}$ is given by

$$H = \begin{bmatrix} \frac{\delta E(\mathbf{w})}{\delta w_0^2} & \frac{\delta E(\mathbf{w})}{\delta w_0 \delta w_1} & \cdots & \frac{\delta E(\mathbf{w})}{\delta w_0 \delta w_m} \\ \frac{\delta E(\mathbf{w})}{\delta w_1 \delta w_0} & \frac{\delta E(\mathbf{w})}{\delta w_1^2} & \cdots & \frac{\delta E(\mathbf{w})}{\delta w_1 \delta w_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta E(\mathbf{w})}{\delta w_m \delta w_1} & \frac{\delta E(\mathbf{w})}{\delta w_m \delta w_2} & \cdots & \frac{\delta E(\mathbf{w})}{\delta w_m^2} \end{bmatrix}$$

Now we can get the new update rule according to Equation 14 as followings,

$$\mathbf{w}^{new} = \mathbf{w}^{old} - H^{-1}\nabla_{\mathbf{w}}E(\mathbf{w}^{old}) \tag{15}$$

Recall that the j^{th} component of $\nabla_{\mathbf{w}}E(\mathbf{w})$ is

$$\frac{\delta E(\mathbf{w})}{\delta w_j} = \sum_{i=1}^{N}(\sigma(\mathbf{w}^T\mathbf{x}_i) - y_i)x_{ij}$$

Also, according to the definition of the Hessian matrix, we have the jk^{th} entry as:

$$\begin{aligned} H_{jk} &= \frac{\delta E(\mathbf{w})}{\delta w_j w_k} \\ &= \frac{\delta \sum_{i=1}^{N}(\sigma(\mathbf{w}^T\mathbf{x}_i) - y_i)x_{ij}}{\delta w_k} \\ &= \frac{\delta \sum_{i=1}^{N}\sigma(\mathbf{w}^T\mathbf{x}_i)x_{ij}}{\delta w_k} - \delta\frac{\sum_{i=1}^{N} y_i x_{ij}}{\delta w_k} \\ &= \frac{\sum_{i=1}^{N}\delta\sigma(\mathbf{w}^T\mathbf{x}_i)x_{ij}}{\delta w_k} \\ &= \frac{\sum_{i=1}^{N}\sigma(\mathbf{w}^T\mathbf{x}_i)(1-\sigma(\mathbf{w}^T\mathbf{x}_i))\delta(\mathbf{w}^T\mathbf{x}_i)x_{ij}}{\delta w_k} \textbf{ (according to equation 2)} \\ &= \sum_{i=1}^{N}\sigma(\mathbf{w}^T\mathbf{x}_i)(1-\sigma(\mathbf{w}^T\mathbf{x}_i))x_{ik}x_{ij} \\ &= \sum_{i=1}^{N} o_i(1-o_i)x_{ik}x_{ij} \end{aligned}$$

We can put the above equation in matrix form:

$$H = \mathbf{X}^T R \mathbf{X} \tag{16}$$

,where R is a diagonal matrix such that $R = \begin{bmatrix} o_1(1-o_1) & 0 & 0 & \dots & 0 \\ 0 & o_2(1-o_2) & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & o_N(1-o_N) \end{bmatrix}$

Combining Equations 15,12 and 16 we get:

$$\mathbf{w}^{new} = \mathbf{w}^{old} - (\mathbf{X}^T R \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{o} - \mathbf{y}) \tag{17}$$

Furthermore, the above equation can be transformed to the following form:

$$\begin{aligned} \mathbf{w}^{new} &= \mathbf{w}^{old} - (\mathbf{X}^T R \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{o} - \mathbf{y}) \\ &= (\mathbf{X}^T R \mathbf{X})^{-1} (\mathbf{X}^T R \mathbf{X} \mathbf{w}^{old} - \mathbf{X}^T (\mathbf{o} - \mathbf{y})) \\ &= (\mathbf{X}^T R \mathbf{X})^{-1} \mathbf{X}^T R z \end{aligned}$$

,where $z = \mathbf{X}\mathbf{w}^{old} - R^{-1}(\mathbf{o} - \mathbf{y})$

Notice that this form is very similar to normal equations that we developed as the analytical solution to linear regression (the least squares solution). Therefore, the Newton-Raphson Method for training logistic regression is also known as the iterative re-weighted least squares (IRLS) algorithm.

5 Regularization

Similar to the linear regression model, logistic regression also suffers from the problem of overfitting. The method to solve this problem is to use regularization. Recall the general framework of regularization we've discussed in the context of ridge regression, we can augment our objective function $E(\mathbf{w})$ with a regularization penalty, such that

$$\widehat{\mathbf{w}}_{reg} = \underset{w \in \mathbb{R}^{m+1}}{\arg\min}\, E(\mathbf{w}) + \lambda ||\mathbf{w}||_q$$

If we use the L1 penalty we obtain following optimization problem :

$$\mathbf{w}_{L1} = \underset{w \in \mathbb{R}^{m+1}}{\arg\min} \sum_{i=1}^{N} -y_i \ln(\sigma(\mathbf{w}^T \mathbf{x}_i)) - (1-y_i) \ln(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)) + \lambda \sum_{i=0}^{m} |w_i|$$

Similarly, if we use the L2 penalty we get the following optimization problem:

$$\mathbf{w}_{L2} = \underset{w \in \mathbb{R}^{m+1}}{\arg\min} \sum_{i=1}^{N} -y_i \ln(\sigma(\mathbf{w}^T \mathbf{x}_i)) - (1-y_i) \ln(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)) + \lambda \sum_{i=0}^{m} w_i^2$$

6 Evaluation

Since the task of logistic regression is classification, we will need a metric to evaluate the performance of our model. Here are two common metrics to evaluate a classification model.

$$accuracy = \frac{\text{number of correctly classified examples}}{\text{total number of examples}} \tag{18}$$

$$recall = \frac{\text{number of correctly classified positive examples}}{\text{total number of positive examples}} \tag{19}$$

7 Loss Function For Logistic Regression:

Recall, that for linear regression we minimized the loss function which was given by the sum of squared errors. What is the underlying loss function that logistic regression optimizes? To answer this question, we see that both the algorithms we presented above for estimating the model parameters, minimized the negative log-likelihood (Equation (8)), which is the loss function for logistic regression. The negative log-likelihood is given as:

$$E(\mathbf{w}) = -\sum_{i=1}^{N} \{y_i \ln(\sigma(\mathbf{w}^T \mathbf{x}_i)) + (1 - y_i) \ln(1 - \sigma(\mathbf{w}^T \mathbf{x}_i))\} \tag{20}$$

which is also known as the *cross-entropy* loss.

8 Reference

1. Andrew Ng, (lecture notes of CS229) Supervised learning
2. `https://en.wikipedia.org/wiki/Sigmoid_function`
3. `https://en.wikipedia.org/wiki/Newton%27s_method`
4. `http://archives.math.utk.edu/visual.calculus/3/newton.6/index.html`
5. `https://en.wikipedia.org/wiki/Lp_space`