Machine Learning Course - CS-433

Logistic Regression

Oct 20, 2016

©Mohammad Emtiyaz Khan 2015

changes by Rüdiger Urbanke 2016

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Logistic regression

Recall that in the previous lecture we discussed what happens if we treat binary classification as regression with lets say $y = 0$ and $y = 1$ as the two possible (target) values and then decide on the label by looking if the predicted value is smaller or larger than 0.5.

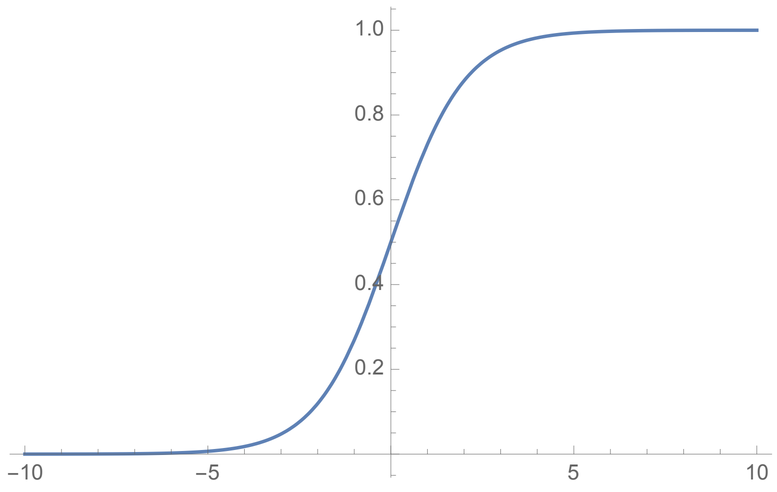
We have also discussed that it might be tempting to interpret the predicted value as probability. But there are problems. The predicted values are in general not in $[0, 1]$ and also very large ($y >> 1$) or very small ($y << 0$) values of the prediction cause problems if we use the squared loss, even though they are an indication of a very confident in the resulting classification.

It is therefore very natural that we *transform* the prediction which can take values in $(-\infty, \infty)$ into a true probability by applying an appropriate function. There are several possible such functions. The *logistic function*

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

is a natural and popular choice, see the next figure.[1]

[1] If you implement this function note that you are applying the exponential function to potentially large (in magnitude) values. This can lead to overflows. One work around is to implement this function where you first check the value of $\mathbf{x}$ and make case distinctions.

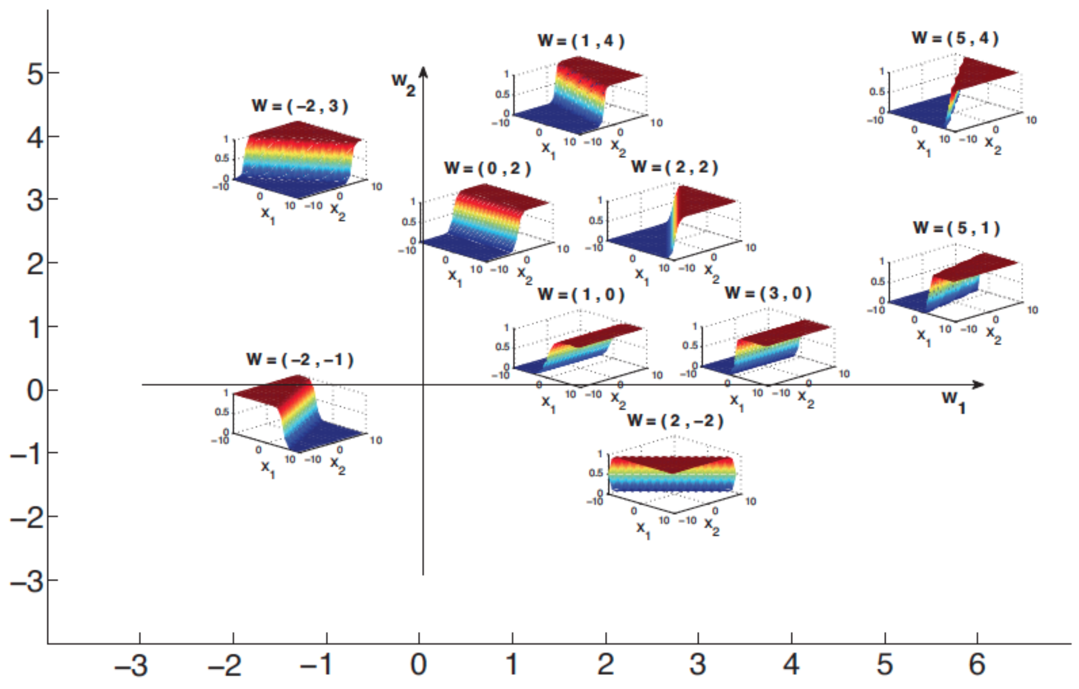


Consider the binary classification case. We proceed as follows. Given a training set S_t we learn a weight vector $\mathbf{w}$ (it remains to discuss how to do this). Given that a "new" feature vector $\mathbf{x}$ we predict the *probability*

$$p(\mathcal{C}_1 \mid \mathbf{x}) = \sigma(\mathbf{x}^T \mathbf{w}),$$
$$p(\mathcal{C}_2 \mid \mathbf{x}) = 1 - \sigma(\mathbf{x}^T \mathbf{w}).$$

Note that we predict a real value (a probability) and not a label. This is the reason it is called *logistic regression.* But typically we use logistic regression as the first step of a classifier. In the second step we quantize the value to a binary value, typically according to wether the predicted probability is smaller or larger than 0.5.
So very large and very small values of y just correspond to probabilities very close to 0 and 1, respectively.

The following figure visualizes the probabilities obtained for a 2-D problem (taken from KPM Chapter 7). More precisely, this is a case with two features and hence two weights that we learn. We see the effect of changing the weight vector on the resulting probability function.



At this point it is hopefully clear how we use logistic regression to do classification. To repeat, given the weight vector $\mathbf{w}$ we predict the probability $p(\mathcal{C}_1 \mid \mathbf{x}) = \sigma(\mathbf{x}^T\mathbf{w})$ and then quantize. What we need to discuss next is how we learn the model, i.e., how we find a good weight vector $\mathbf{w}$ given some training set S_t.

Training

As always we assume that we have our training set S_t, consisting of iid samples $\{(y_n, \mathbf{x}_n)\}_{n=1}^N$, sampled according to a fixed but unknown distribution $\mathcal{D}$. Exploiting that the samples $(y_n, \mathbf{x}_n)$ are independent, the probability of $\mathbf{y}$ (vector of all labels) given $\mathbf{X}$ (matrix of all inputs) and $\mathbf{w}$ (weight vector) has a simple product form:

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}, \mathbf{w}) &= \prod_{n=1}^{N} p(y_n|\mathbf{x}_n) \\ &= \prod_{n:y_n=\mathcal{C}_1} p(y_n = \mathcal{C}_1|\mathbf{x}_n) \prod_{n:y_n=\mathcal{C}_2} p(y_n = \mathcal{C}_2|\mathbf{x}_n) \end{aligned}$$

A better way to write this is to use the coding $y_n \in \{0, 1\}$ where we assume that we encode $\mathcal{C}_1$ by the target value 1 and $\mathcal{C}_2$ by the target value 0. With this notation we get

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}, \mathbf{w}) &= \prod_{n:y_n=\mathcal{C}_1} p(y_n = \mathcal{C}_1|\mathbf{x}_n) \prod_{n:y_n=\mathcal{C}_2} p(y_n = \mathcal{C}_2|\mathbf{x}_n) \\ &= \prod_{n=1}^{N} \sigma(\mathbf{x}^T\mathbf{w})^{y_n}[1 - \sigma(\mathbf{x}^T\mathbf{w})]^{1-y_n} \end{aligned}$$

It is convenient to take the logarithm of this probability to bring it into an even simpler form. In addition we add a minus sign to the expression. In this way our objective will be to minimize the resulting cost function (rather than maximizing it) as we have done this in the past for other cost functions.

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= -\sum_{n=1}^{N} y_n \ln \sigma(\mathbf{x}_n^\top \mathbf{w}) + (1-y_n)\ln[1-\sigma(\mathbf{x}_n^\top \mathbf{w})] \\ &= \sum_{n=1}^{N} \ln[1+\exp(\mathbf{x}_n^\top \mathbf{w})] - y_n \mathbf{x}_n^\top \mathbf{w}.\end{aligned}$$

In the last step we have used the specific form of the logistic function $\sigma(x)$ to bring the cost function into a nice form.

Maximum likelihood criterion

Recall what we did so far. Under the assumption that the samples are independent we have written down the logarithm of the probability that we observe the label vector $\mathbf{y}$ given the inputs $\mathbf{X}$ and a particular choice of weights $\mathbf{w}$. It is now natural that we choose that weight vector $\mathbf{w}$ that maximizes this probability.
Equivalently, we choose that weight that maximizes the log of this probability, or mimizes $\mathcal{L}(\mathbf{w})$ (recall that we added negative sign in order to arrive at a minimization of the cost function). This is called

the *maximum-likelihood criterion.*[2]
So in order to get the best weight vector, call it $\mathbf{w}^*$, we have to perform the following opimization.

$$\mathbf{w}^* = \text{argmin}_{\mathbf{w}} \mathcal{L}(\mathbf{w}).$$

Condition of optimality

As we discussed, we want to minimize $\mathcal{L}(\mathbf{w})$. Therefore let us look at the stationary points of this function by computing the gradient and setting it to zero, $\nabla \mathcal{L}(\mathbf{w}) = 0$.
Note that

$$\frac{d \ln[1 + \exp(x)]}{dx} = \sigma(x).$$

Therefore

$$\begin{aligned} \nabla \mathcal{L}(\mathbf{w}) &= \sum_{n=1}^{N} \mathbf{x}_n (\sigma(\mathbf{x}_n^\top \mathbf{w}) - y_n) \\ &= \mathbf{X}^T [\sigma(\mathbf{X}\mathbf{w}) - \mathbf{y}]. \end{aligned}$$

In the last step we have used the following convenient notation. Recall that the matrix $\mathbf{X}$ has N

[2]There is perhaps some chance of confusion regarding nomenclature here. If we think of $\mathbf{w}$ as fixed, then $p(\mathbf{y}|\mathbf{X}, \mathbf{w})$ is the *posterior* of $\mathbf{y}$ given the input/observations $\mathbf{X}$. But we are interested in $p(\mathbf{y}|\mathbf{X}, \mathbf{w})$ as a function of $\mathbf{w}$ and not $\mathbf{y}$ and since $\mathbf{w}$ appears on the right in this conditional probability it is a likelihood, and $\mathcal{L}(\mathbf{w})$ is the log-likelihood of $\mathbf{w}$. This explains why we are calling this the maximum (log)likelihood criterion.

rows that correspond to the N samples and each row corresponds to one particular input. Further, $\mathbf{y}$ is the column vector of length N which represents the N labels corresponding to the N samples.[3] Therefore, $\mathbf{Xw}$ is a column vector of length N. The expression $\sigma(\mathbf{Xw})$ means that we apply the function σ to each of the N components of $\mathbf{Xw}$. In this manner we can express the gradient in a compact manner.
There are no closed-form solutions for this equation. Let us therefore discuss how to solve this equation in an iterative fashion by using gradient descent or the Newton method.

Convexity

Since we are planning to iteratively minimize our cost function, it is good to know that this cost function is convex.

Lemma 0.1. *The log-likelihood*

$$\mathcal{L}(\mathbf{w}) = \sum_{n=1}^{N} \ln[1+\exp(\mathbf{x}_n^\top \mathbf{w})] - y y_n \mathbf{x}_n^\top \mathbf{w} -$$

is convex in the weights $\mathbf{w}$.

[3] This notation is a little bit unfortunate – we have the expression $\mathbf{x}^T\mathbf{w}$ if we are looking at the prediction for a single sample but $\mathbf{Xw}$ if we are looking at all samples together.

Proof. Recall that the sum of any number of (strictly) convex functions is (strictly) convex. Note that $\mathcal{L}(\mathbf{w})$ is the sum of $2N$ functions. N of them are linear in $\mathbf{w}$ and a linear function is convex. Therefore it suffices to show that the other N functions are convex as well. Let us consider one of those. It has the form $\log[1 + \exp(\mathbf{x}_n^\top \mathbf{w})]$. Note that $\ln(1+\exp(x))$ is convex (it has first derivative $\sigma(x)$ and second derivative $\exp(x)/(1+\exp(x))^2$, which is non-negative). Hence $\ln[1 + \exp(\mathbf{x}_n^\top \mathbf{w})]$ is the composition of two convex functions (a linear function composed with $\ln(1+\exp(x))$, and is hence convex. □

Note: Alternatively, to prove that a function is convex (strictly convex) we can check that the Hessian (matrix consisting of second derivatives) is positive semi-definite (positive definite).

Gradient descent

To be completed!

Newton's method

Hessian of the Log-Likelihood

We will use the following fact:

$$\frac{\partial \sigma(t)}{\partial t} = \sigma(t)[1 - \sigma(t)]$$

Taking the derivative of the gradient we get the Hessian,

$$\mathbf{H}(\mathbf{w}) := -\frac{\partial \mathbf{g}(\mathbf{w})}{\partial \mathbf{w}^\top} = \widetilde{\mathbf{X}}^\top \mathbf{S} \widetilde{\mathbf{X}}$$

where $\mathbf{S}$ is a $N \times N$ diagonal matrix with diagonals

$$S_{nn} = \sigma(\widetilde{\mathbf{x}}_n^\top \mathbf{w})[1 - \sigma(\widetilde{\mathbf{x}}_n^\top \mathbf{w})].$$

Is the negative of the log-likelihood *strictly* convex?

Newton's Method

Gradient descent uses only first-order information and takes steps in the direction of the gradient. Newton's method uses second-order information and takes steps in the direction that minimizes a quadratic approximation.

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \gamma_t \mathbf{H}_t^{-1} \mathbf{g}_t$$

where $\mathbf{g}_t$ is the gradient.

Computational complexity

Compare the computational complexity of least-squares and Newton's method.
Newton's method is equivalent to solving many least-squares problems.

Penalized Logistic Regression

The cost-function can be unbounded when the data is linearly separable.
For a well-defined problem, we will regularize.

$$\min_{\mathbf{w}} -\sum_{n=1}^{N} \ln p(y_n|\mathbf{x}_n^\top, \mathbf{w}) + \lambda \sum_{d=1}^{D} \mathbf{w}_d^2$$

Additional notes

Derivation of Newton's method

The second-order approximation of a function is given as follows:

$$\mathcal{L}_Q(\mathbf{w}) := \mathcal{L}(\mathbf{w}^{(t)}) + \mathbf{g}_t^\top(\mathbf{w} - \mathbf{w}^{(t)}) + \tfrac{1}{2}(\mathbf{w} - \mathbf{w}^{(t)})^\top \mathbf{H}_t(\mathbf{w} - \mathbf{w}^{(t)})$$

The minimum of $\mathcal{L}_Q$ is at $\mathbf{w}^{(t)} - \mathbf{H}_t^\top \mathbf{g}_t$. A conservative option is to take a small step in this direction using step-size γ_t, which is the step used in Newton's method.

Set γ_t using line search, e.g. the Armijo rule. See Section 8.3.2 of Kevin Murphy's book. A good implementation can be found on page 29 of Bertsekas book “Non-linear programming”.

Iterative Recursive Least-Squares (IRLS)

(IRLS) expresses Newton’s method with $\gamma_t = 1$ as a sequence of least-squares problems. Below is the derivation and pseudo code.

$$\begin{aligned}
\mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} - \gamma_t \mathbf{H}_t^{-1}\mathbf{g}_t && (1)\\
&= \mathbf{w}^{(t)} - (\widetilde{\mathbf{X}}^\top \mathbf{S}_t \widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^\top(\boldsymbol{\sigma}_t - \mathbf{y}) && (2)\\
&= (\widetilde{\mathbf{X}}^\top \mathbf{S}_t \widetilde{\mathbf{X}})^{-1}[(\widetilde{\mathbf{X}}^\top \mathbf{S}_t \widetilde{\mathbf{X}})\mathbf{w}^{(t)} - \widetilde{\mathbf{X}}^\top(\boldsymbol{\sigma}_t - \mathbf{y})]\\
&= (\widetilde{\mathbf{X}}^\top \mathbf{S}_t \widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^\top \mathbf{X}_t[\widetilde{\mathbf{X}}\mathbf{w}^{(t)} + \mathbf{S}_t^{-1}(\mathbf{y} - \boldsymbol{\sigma}_t)]\\
&= (\widetilde{\mathbf{X}}^\top \mathbf{S}_t \widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^\top \mathbf{X}_t \mathbf{z}_t && (3)
\end{aligned}$$

where $\mathbf{z}_t = \widetilde{\mathbf{X}}\mathbf{w}^{(t)} + \mathbf{S}_t^{-1}(\mathbf{y} - \boldsymbol{\sigma}_t)$.

```
for t = 1:maxIters
    sig = sigmoid(tX*weights);
    s = sig.*(1-sig);
    z = tX*weights + (y-sig)./s;
    weights = weightedLeastSquares(z,tX,s);
end
```

Quasi-Newton

Read about L-BFGS in Section 8.3.5 of Kevin Murphy’s book. The key idea is to approximate **H** usign a diagonal and a low-rank matrix.

ToDo

1. Practice to derive the cost function using maximum likelihood estimation.
2. Understand the normal equation.
3. Understand the interpretation of log-odds (JWHT Chapter 3).
4. Learn to prove convexity using the positive-definite property of the Hessian.
5. Implement Newton's method (part of next week's lab).
6. Understand the relationship of Newton's Method with IRLS.