Machine Learning Course - CS-433

Logistic Regression

Oct 20, 2016

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

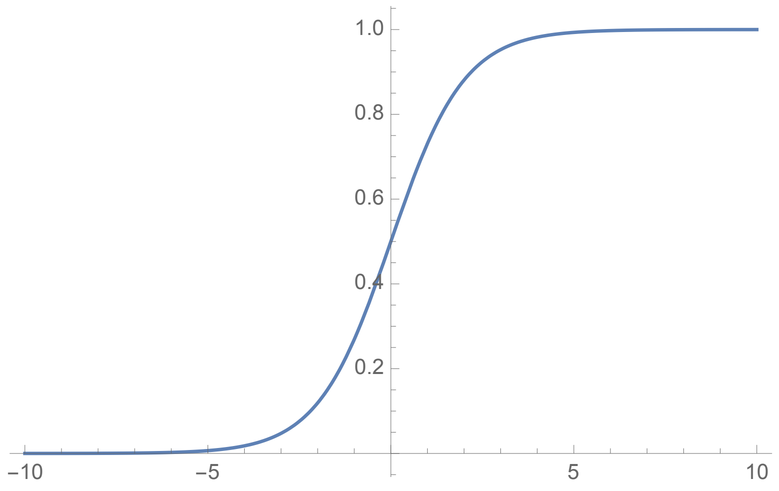
Logistic regression

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 is 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 indicate that we are 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



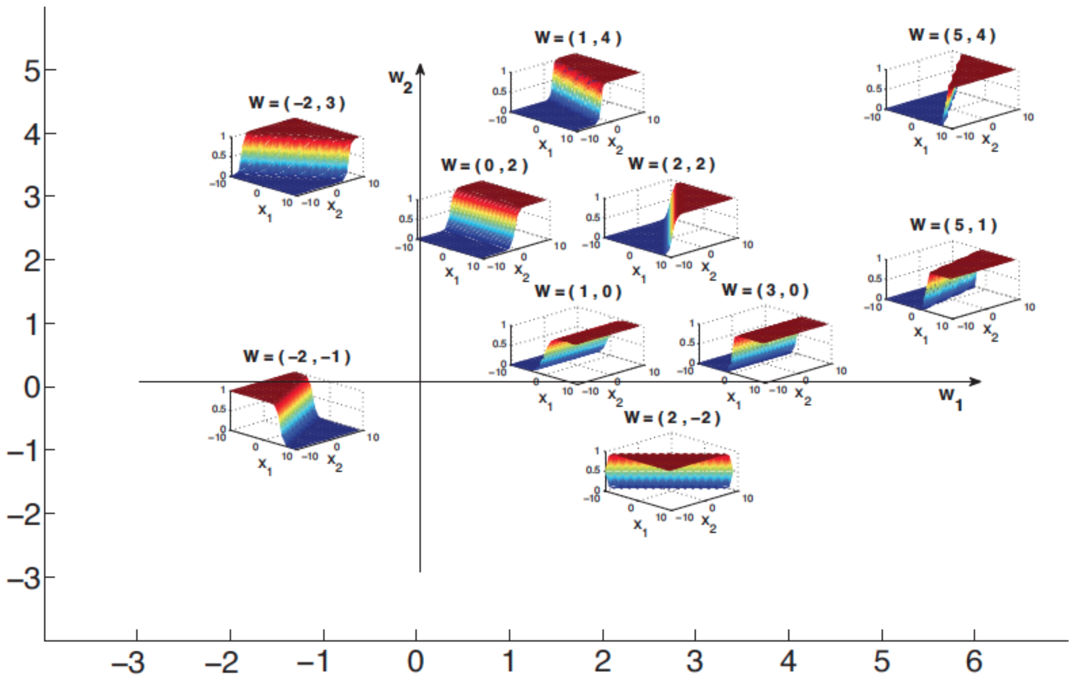
Consider the binary classification case. We proceed as follows. Given a training set S_t we learn a weight vector $\mathbf{w}$ (we will discuss how to do this very soon). Given a "new" feature vector $\mathbf{x}$, we predict the (posterior) *probability* of the two class labels given $\mathbf{x}$ by means of

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

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.

can lead to overflows. One work around is to implement this function by first checking the value of x and by treating large (in magnitude) values separately.

So very large and very small (large negative) values of $\mathbf{x}^T\mathbf{w}$ 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 of the class label $\mathcal{C}_1$ to be $p(\mathcal{C}_1 \mid \mathbf{x}) = \sigma(\mathbf{x}^\top\mathbf{w})$ 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.

A word about notation

In the beginning of this course we started with an arbitrary feature vector $\mathbf{x}$. We then discussed that often it is useful to add the constant 1 to this feature vector and we called the resulting vector $\widetilde{\mathbf{x}}$. We also discussed that often it is useful to add further features and we called then the resulting vector $\phi(\mathbf{x})$. *We will assume from now on that the vector* $\mathbf{x}$ *always contains the constant term as well as any further features we care to add.* This will save us from a flood of notation. Note that in particular for the logistic regression it is crucial that we have the constant term contained in $\mathbf{x}$.

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

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

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). This is consistent with our previous examples, where we always minimized the cost function. We call the resulting cost function $\mathcal{L}(\mathbf{w})$,

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

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 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]

[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

So in order to get the best weight vector, call it $\mathbf{w}^\star$, we have to perform the following opimization,

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

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,setting it to zero, and solving for $\mathbf{w}$. Note that

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

Therefore

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

In the last step we have used the following convenient notation. Recall that the matrix $\mathbf{X}$ has N rows that correspond to the N samples and each row corresponds to one particular input. Further,

$\mathbf{y}$ 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. This explains why we are calling this the maximum likelihood criterion.

$\mathbf{y}$ 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 is no closed-form solution 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. *The log-likelihood*

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

is convex in the weight vector $\mathbf{w}$*.*

Proof. Recall that the sum (with non-negative weights) of any number of (strictly) convex functions is (strictly)

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

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 the non-negative second derivative

$$\frac{d^2 \ln(1+\exp(x))}{dx^2} = \frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x) \geq 0. \quad (1)$$

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 therefore convex. $\square$

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

As we have done for other cost functions, we can apply a (stochastic) gradient descent algorithm to minimize our cost. E.g. for the batch version we can implement the update equation

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma^{(t)} \nabla \mathcal{L}(\mathbf{w}^{(t)}),$$

where $\gamma^{(t)} > 0$ is the step size and $\mathbf{w}^{(t)}$ is the sequence of weight vectors.

Newton's method

The gradient method is a *first-order* method, i.e., it only uses the gradient (the first derivative). We get a more powerful optimization algorithm if we use also the second order terms. Of course there is a trade-off. On the one hand we need fewer steps to converge if we use second order terms, on the other hand every iteration is more costly. Let us describe now a scheme that also makes use of second order terms. It is called *Newton's method.*

Hessian of the Log-Likelihood

Let us compute the *Hessian* of the cost function $\mathcal{L}(\mathbf{w})$, call it $\mathbf{H}(\mathbf{w})$. What is the Hessian? If $\mathbf{w}$ has D components then this is the $D \times D$ symmetric matrix with entries

$$\mathbf{H}_{i,j} = \frac{\partial^2 \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}_i \partial \mathbf{w}_j}.$$

Recall that the costfunction $\mathcal{L}(\mathbf{w})$ is a sum over N terms, all of the same form. So let us first compute the Hessian corresponding to one such term. We already computed the *gradient* of one such term and got

$$\mathbf{x}_n(\sigma(\mathbf{x}_n^T\mathbf{w}) - y_n).$$

Recall, that this gradient is a vector of length D (the dimension of the feature vector $\mathbf{x}$ and hence also the dimension of the weight vector) where the i-th component is the derivative of $\mathcal{L}(\mathbf{w})$ with respect to $\mathbf{w}_i$. If you look at the above expression you see that this gradient is equal to $\mathbf{x}$ (a vector) times the scalar $(\sigma(\mathbf{x}_n^T\mathbf{w}) - y_n)$. Note that $\mathbf{x}$ does not depend on $\mathbf{w}$ and neither does y_n. The only dependence is in the term $\sigma(\mathbf{x}_n^T\mathbf{w})$. Therefore, the Hessian associated to one term will be

$$\mathbf{x}\nabla\sigma(\mathbf{x}_n^T\mathbf{w}).$$

We have already seen that $\sigma'(x) = \sigma(x)(1-\sigma(x))$. Therefore, by the chain rule one such term gives rise to the Hessian

$$\mathbf{x}\mathbf{x}^T\sigma(\mathbf{x}_n^T\mathbf{w})(1 - \sigma(\mathbf{x}_n^T\mathbf{w})).$$

It remains to do the sum over all N samples. Rather than just summing, let us put this again in a compact form by using the data matrix $\mathbf{X}$. We get

$$\mathbf{H}(\mathbf{w}) = \mathbf{X}^\top\mathbf{S}\mathbf{X},$$

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

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

Newton's Method

Gradient descent uses only first-order information and takes steps in the direction opposite to the gradient. This makes sense since the gradient points in the direction of increasing function values and we want to *minimize* the function.
Newton's method uses second-order information and takes steps in the direction that minimizes a quadratic approximation. More precisely, it approximates the function locally by a quadratic form and then moves in the direction where this quadratic form has its minimum. The update equation is of the form

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \gamma^{(t)}(\mathbf{H}^{(t)})^{-1}\nabla\mathcal{L}(\mathbf{w}^{(t)}).$$

Where does this update equation come from?
Recall that the Taylor series approximation of a function (up to second order terms) around a point $\mathbf{w}^\star$ has the form.

$$\begin{aligned}\mathcal{L}(\mathbf{w}) \approx \mathcal{L}(\mathbf{w}^\star) + (\nabla\mathcal{L}(\mathbf{w}^\star)^\top(\mathbf{w} - \mathbf{w}^\star) + \\ + \tfrac{1}{2}(\mathbf{w} - \mathbf{w}^\star)^\top\mathbf{H}(\mathbf{w}^\star)(\mathbf{w} - \mathbf{w}^\star).\end{aligned}$$

The right-hand side is a *local approximation* of $\mathcal{L}(\mathbf{w})$ but it is of course not exact since we left out higher-order terms. But assume that we take the right-hand side to be exact representation of our cost function. We want to minimize the function. So let us look where the function defined by the right-hand side takes its minimum value. If we think that the approximation is reasonably good, then it makes sense to move the new weight vector to the position of this minimum.
Let us take the gradient of the right hand side and set it to zero. We get

$$\nabla\mathcal{L}(\mathbf{w}^*) + \mathbf{H}(\mathbf{w}^*)(\mathbf{w} - \mathbf{w}^*) = 0.$$

Solving for $\mathbf{w}$ gives us $\mathbf{w} = \mathbf{w}^* - \mathbf{H}(\mathbf{w}^*)^{-1}\nabla\mathcal{L}(\mathbf{w}^*)$. This corresponds exactly to the stated update equation, except that in this update we have an extra step size γ. Why do we need this factor?
Recall that the right-hand side is only an approximation. Caution therefore dictates that we only move part of the way to the indicated minimum.

Penalized Logistic Regression

Although the cost-function for logistic regression is lower bounded by 0 we get issues if the data is linearly separable. In this case there is no finite-weight vector $\mathbf{w}$ which gives us this minimum cost

function and if we continue to run the optimization the weights will tend to infinity.
To avoid this problem, as for standard regression problems, we can add a penality term. E.g., we consider the cost function

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

Additional notes

Step size in Newton's method

Set $\gamma^{(t)}$ 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".

Computational complexity and iterative recursive least-squares (IRLS)

Newton's method is equivalent to solving many least-squares problems. (IRLS) expresses Newton's method with $\gamma^{(t)} = 1$ as a sequence of least-squares problems. Below is the derivation.

$$\begin{aligned}
\mathbf{w}^{(t+1)} &:= \mathbf{w}^{(t)} - \gamma^{(t)}(\mathbf{H}^{(t)})^{-1}\nabla\mathcal{L}(\mathbf{w}^{(t)}) \\
&= \mathbf{w}^{(t)} - (\mathbf{X}^\top\mathbf{S}^{(t)}\mathbf{X})^{-1}\mathbf{X}^\top(\boldsymbol{\sigma}^{(t)} - \mathbf{y}) \\
&= (\mathbf{X}^\top\mathbf{S}^{(t)}\mathbf{X})^{-1}[(\mathbf{X}^\top\mathbf{S}^{(t)}\mathbf{X})\mathbf{w}^{(t)} - \mathbf{X}^\top(\boldsymbol{\sigma}^{(t)} - \mathbf{y})] \\
&= (\mathbf{X}^\top\mathbf{S}^{(t)}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{X}^{(t)}[\mathbf{X}\mathbf{w}^{(t)} + (\mathbf{S}^{(t)})^{-1}(\mathbf{y} - \boldsymbol{\sigma}^{(t)})] \\
&= (\mathbf{X}^\top\mathbf{S}^{(t)}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{X}^{(t)}\mathbf{z}^{(t)}
\end{aligned}$$

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

Quasi-Newton

Read about L-BFGS in Section 8.3.5 of Kevin Murphy's book. The key idea is to approximate $\mathbf{H}$ using a diagonal and a low-rank matrix.

ToDo

1. Practice to derive the cost function using maximum likelihood estimation.
2. Understand the normal equation (condition of optimality).
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.