50.021 -AI

Alex

Week 02: Classification by logistic regression

Logistic regression

Lets come back to classification.

Given an input space $\mathcal{X}$, the goal is to predict for every sample $x \in \mathcal{X}$ to which class it belongs. For 2 class classification, the goal is: to predict in the output space $\mathcal{Y} = \{-1, +1\}$ or $\mathcal{Y} = \{0, 1\}$.

Can be generalized to C classes, then $\mathcal{Y} = \{0, 1, 2, 3, \ldots, C-1\}$. Focus on 2 classes here.

Focus on a method which is related to linear regression and neural networks.

Linear regression has unbounded values, we need values in $\{-1, +1\}$

$$g(x) = w \cdot x$$

We had before the following idea for a classification mapping $f(x) = sign(g(x))$. So if $f(x) > 0$, then we predict the label $+1$, otherwise we predict -1 ($f(x) = 0$ is a tie case).

Lets derive a thought from this:

- The classification switches at the set of points x such that $f(x) = 0$.
- Intuition: for points x with $f(x) \approx 0$, we are not sure about the prediction.

  the larger $|f(x)|$, the more confident one should be to predict a class label.
- $f(x) >> 0$ large positive, we should be confident about the prediction $+1$
- $f(x) << 0$ large negative, we should be confident about the prediction -1

  Can we encode this confidence as a probability?

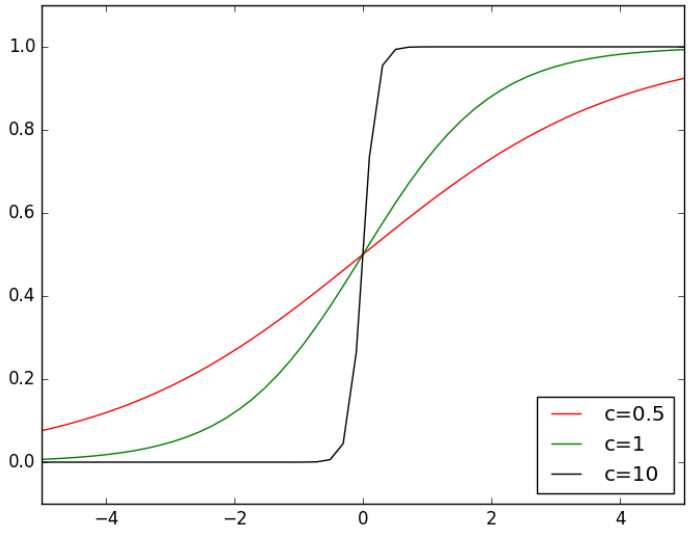
Lets map $(-\infty, +\infty)$ onto $(0,1)$ such that 0 gets mapped onto 0.5:

$$s(a) = \frac{\exp(a)}{1+\exp(a)} = \frac{1}{\exp(-a)+1}\frac{\exp(a)}{\exp(a)} = \frac{1}{\exp(-a)+1}$$

exponential function

$$\lim_{a\to-\infty} s(a) = \frac{\exp(-\infty)}{1+\exp(-\infty)} = \frac{0}{1+0} = 0$$

$$\lim_{a\to+\infty} s(a) = \frac{1}{\exp(-\infty)+1} = \frac{1}{0+1} = 1$$



$s(ca) = \frac{\exp(ca)}{1+\exp(ca)} = \frac{1}{\exp(-ca)+1}$ for different values of c.

The convergence of $s(a \to -\infty) \to 0, s(a \to +\infty) \to 1$ shows that: $s(a)$ can be interpreted as a probability, expressing confidence for $P(Y = +1|X = x)$

interpretation of probability

s **is called the logistic function**

Definition: Logistic function

$$s(a) = \frac{\exp(a)}{1+\exp(a)} = \frac{1}{\exp(-a)+1}\frac{\exp(a)}{\exp(a)} = \frac{1}{\exp(-a)+1}$$

Definition: Logistic regression model

The logistic regression model is the concatenation of the linear/affine mapping $f_w(x) = w \cdot x$ with the logistic function $s(a)$.

$$h(x) = s(f_w(x)) = \frac{\exp(w \cdot x)}{1 + \exp(w \cdot x)} = \frac{1}{\exp(-w \cdot x) + 1}$$

Its output $h(x) \in (0, 1)$ can be interpreted for 2 classes as $P(Y = +1 | X = x)$ – the confidence that sample x has classification label $y = +1$.[1]

[1] Why $s(a)$ encodes a conditional probability $P(Y = +1 | X = x)$ and not a joint probability $P(Y = +1, X = x)$? – bcs the latter would have $P(Y = +1, X = x) + P(Y = -1, X = x) = f(X = x)$ – the density of the samples x which is nowhere modeled in it

We need to define a loss function for this model. Model outputs a probability, so one can use maximum likelihood principle.

Definition: The loss function for logistic regression

The loss for logistic regression to be **minimized** is the negative log likelihood of the probability of observing all labels $y_i \in \{0, 1\}$ for given data points x_i

$$\text{argmin}_w (-1) \cdot \sum_{i=1}^{n} y_i \log\left(h(x_i)\right) + (1 - y_i) \log\left(1 - h(x_i)\right)$$

Looks complicated, but all we need is:

- the maximum likelihood principle
- $h(x) = P(Y = +1 | X = x)$
- $P(Y = y | X = x) = h(x)^y (1 - h(x))^{1-y}$ given that $y \in \{0, +1\}$ (otherwise transform $y_i \in \{-1, +1\}$ to $(y_i + 1)/2 \in \{0, 1\}$)
- independence: $P(Y_1 = y_1, Y_2 = y_2, \ldots, Y_n = y_n | x_1, x_2, ..., x_n) = P(Y_1 = y_1 | x_1) \cdot P(Y_2 = y_2 | x_2) \cdot \ldots \cdot P(Y_n = y_n | x_n)$
- $L(w) = P(Y_1 = y_1, Y_2 = y_2, \ldots, Y_n = y_n | x_1, x_2, ..., x_n) = \prod_{i=1}^{n} P(Y_i = y_i | x_i)$
- goal: want to maximize the probability of observing all labels y_i for given data points x_i, so maximize $L(w)$
- $w^* = \text{argmin} - log(L(w))$

Recap: maximum likelihood:

Suppose one has a dataset $D_n = (z_1, \ldots, z_n)$ and wants to fit *the parameter* w of a probability model P. P takes $(z_1, \ldots, z_n)$ as input and produces a probability $P(z_1, \ldots, z_n | w) \in (0, 1)$ for observing the dataset $(z_1, \ldots, z_n)$.

The maximum likelihood principle states: Given D_n, choose the parameter w such that the probability of observing the data we have is maximized, that is:

$$w^* = \text{argmax}_w P(z_1, \ldots, z_n | w)$$

Note: maximum a-posteriori, fully bayesian methods offer often better ways to fit models.

Derivation of the loss function:

How to apply this idea? We want to maximize the joint probability of observing all the labels y_i given all the data points x_i:

$$w^* = \text{argmax}_w P(Y_1 = y_1, Y_2 = y_2, \ldots, Y_n = y_n | x_1, x_2, ..., x_n)$$

How to arrive at this ?We have

- a dataset $D_n = ((x_1, y_1), \ldots, (x_n, y_n))$.
- $h(x) = P(Y = +1 | X = x)$
- For likelihood we need an expression of type $P(Y = y | X = x)$. For a pair (x, y) with $y \in \{-1, +1\}$:

$$\begin{aligned} P(Y = y | X = x) &= P(Y = +1 | X = x)^{(y+1)/2} \cdot P(Y = -1 | X = x)^{1-(y+1)/2} \\ &= h(x)^{(y+1)/2} (1 - h(x))^{1-(y+1)/2} \quad \text{for } y \in \{-1, 1\} \end{aligned}$$

Check that for $y = -1$ and for $y = +1$ it returns the correct values $P(Y = -1 | X = x)$ and $P(Y = +1 | X = x)$.

Note that $(y_i + 1)/2$ does only mapping the labels onto $\{0, 1\}$. If we would assume $y \in \{0, 1\}$, then this gets simpler:

$$\begin{aligned} P(Y = y | X = x) &= P(Y = +1 | X = x)^{y} \cdot P(Y = 0 | X = x)^{1-y} \\ &= h(x)^{y} (1 - h(x))^{1-y} \quad \text{for } y \in \{0, 1\} \end{aligned}$$

From now on we will assume $y_i \in \{0, 1\}$

if we **assume that all samples are statistically independent**, that is $P((x_1, y_1), \ldots, (x_n, y_n)) = P(x_1, y_1) \cdot P(x_2, y_2) \cdot \ldots \cdot P(x_n, y_n)$, then we can get:

$$\begin{aligned} & P(Y_1 = y_1, Y_2 = y_2, \ldots, Y_n = y_n | x_1, x_2, ..., x_n) \\ = P(Y_1 = y_1 | x_1, \ldots, x_n) \cdot P(Y_2 = y_2 | x_1, \ldots, x_n) \cdot \ldots \cdot P(Y_n = y_n | x_1, \ldots, x_n) \\ & = P(Y_1 = y_1 | x_1) \cdot P(Y_2 = y_2 | x_2) \cdot \ldots \cdot P(Y_n = y_n | x_n) \\ = h(x_1)^{y_1} (1 - h(x_1))^{1-y_1} \cdot h(x_2)^{y_2} (1 - h(x_2))^{1-y_2} \cdot \ldots \cdot h(x_n)^{y_n} (1 - h(x_n))^{1-y_n} \end{aligned}$$

Result: applying maximum likelihood, to find the parameter w, such that the probability $P(Y_1 = y_1, Y_2 = y_2, \ldots, Y_n = y_n | x_1, x_2, ..., x_n)$ is maximized is the same as maximizing the term

$$L(w) = \prod_{i=1}^{n} h(x_i)^{y_i} (1 - h(x_i))^{1-y_i}$$

with respect to parameter w.

This is a product –difficult to optimize using gradients (product rule), instead we can maximize the logarithm of this term.

- Logarithms turns products into sums.

$$log(\prod_i a_i) = \sum_i log(a_i)$$

- Logarithms preserve points minizing or maximizing a function: If a point w is a maximizer of a $L(w)$, then it is also a maximizer of $\log L(w)$, because a logarithm is a strictly monotonically growing function . That is:

$$w^* = \text{argmax} L(w)$$
$$\text{so: } L(w^*) > L(w) \Leftrightarrow \log L(w^*) > \log L(w)$$

The final step: Maximizing a function f is same as minimizing -1 times this function f, so in the end: we can minimize the negative logarithm of above likelihood, in short: neg-log-likelihood. Thus our **loss function** will be the negative logarithm of the likelihood.

Goal:

$$\begin{aligned}
w^* &= \text{argmin}_w - \log\left(L(w)\right) \\
&= \text{argmin}_w - \log\left(\prod_{i=1}^{n} h(x_i)^{y_i}(1-h(x_i))^{1-y_i}\right) \\
&= \text{argmin}_w (-1) \cdot \sum_{i=1}^{n} \log\left(h(x_i)^{y_i}(1-h(x_i))^{1-y_i}\right) \\
&= \text{argmin}_w (-1) \cdot \sum_{i=1}^{n} \log\left(h(x_i)^{y_i}\right) + \log\left((1-h(x_i))^{1-y_i}\right) \\
&= \text{argmin}_w (-1) \cdot \sum_{i=1}^{n} y_i \log\left(h(x_i)\right) + (1-y_i)\log\left(1-h(x_i)\right)
\end{aligned}$$

Using the gradient of the loss

The gradient of the loss function is simple. We will use

$$\frac{\partial log(s(a))}{\partial a} = 1 - s(a)$$
$$\frac{\partial log(1-s(a))}{\partial a} = -s(a)$$

and we know $h(x) = s(w \cdot x)$

The gradient with respect to w will be:

$$\begin{aligned}\nabla_w L &= \nabla_w \left((-1) \cdot \sum_{i=1}^{n} y_i \log(s(w \cdot x_i)) + (1 - y_i) \log\left(1 - s(w \cdot x_i)\right) \right) \\ &= (-1) \cdot \sum_{i=1}^{n} y_i (1 - s(w \cdot x_i)) x_i + (1 - y_i) \cdot (-1) s(w \cdot x_i) x_i \\ &= (-1) \cdot \sum_{i=1}^{n} y_i x_i - y_i s(w \cdot x_i) x_i - s(w \cdot x_i) x_i + y_i s(w \cdot x_i) x_i\end{aligned}$$

Two terms cancel out here and we arrive at

$$\nabla_w L = \sum_{i=1}^{n} x_i (s(w \cdot x_i) - y_i) = \sum_{i=1}^{n} x_i (h(x_i) - y_i)$$

The gradient is the sum of data points x_i weighted by differences between the predicted value $h(x_i) = s(w \cdot x_i)$ and the true value $y_i \in \{0, 1\}$.

If both classes in the data can be perfectly separated by a linear mapping, then optimization will try to make the weights w to go to infinity. Why? $h(x) = s(wx)$ converges to $-1, +1$ for very large values of wx. so making $P(Y = y|X = x)$ to 1 for all data points – requires to push $f(x) = wx$ to very large values for all data points. This can be done by upscaling w.

What is missing here? A bias term.

$$f_{w,b}(x) = s(wx + b)$$

Can be approximately included by augmenting all your datapoints by adding an extra dimension which is constant :

$$\begin{aligned}x_i = (x_i^{(1)}, \ldots, x_i^{(d)}) &\mapsto \hat{x}_i = (x_i^{(1)}, \ldots, x_i^{(d)}, 1) \\ (w, w^{(d+1)}) \cdot \hat{x}_i &= w \cdot x_i + w^{(d+1)} \cdot 1 \sim wx + b\end{aligned}$$

The only caveat here: any regularization term on w now has effect on the bias term, too. Zero-meaning the features makes this effect less harmful.

Demo:

One can observe oscillations for too high learning rates, even for a simple 2 by 2 point dataset.

Can use again polynomial features.

In class coding task:

- Complete the implementation of sgdLogReg. This is the stochastic gradient descent version of gdLogReg (defined in learn.py).

- Test your implementation by calling txor_more with *method* $='$ *sgdLog*$'$. Compare to the behavior of the batch gradient descent *method* $='$ *gdLog*$'$ implementation. Consider the effects of step_size and max_iter for polynomial orders 3 and 5.