

Kernel Logistic Regression: Everything You Need To Know

For classification tasks, it is often the case that the optimal decision boundary is a nonlinear function. In these cases, linear classifiers can explore the data's separability in an alternative feature space using kernel functions. A kernel function $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ map two points in the input space to a real number, and correspond to the inner product of two feature maps in some unique feature space (known as a Reproducing Kernel Hilbert Space). That is,

$$K(x, x') = \langle \Phi(x), \Phi(x') \rangle$$

It turns out that the solution to many learning problems can be written as a linear combination of the kernel function at the training set points. That is, for the problem of minimizing the (convex and continuous) empirical risk,

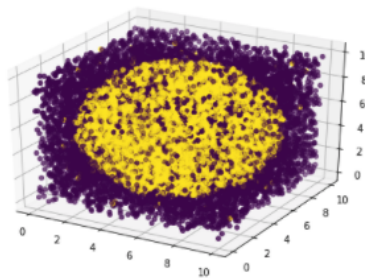
$$\inf_{f \in \mathcal{H}} \hat{\mathcal{E}}(f) + \lambda \|f\|_{\mathcal{H}}^2,$$

the optimal function \hat{f} can be written as

$$\hat{f}(x) = \sum_i^n K(x, x_i) c_i$$

What is even more interesting is that we can discover the weights c using gradient descent without ever invoking the feature map Φ that corresponds to the kernel function K . Indeed in some cases, Φ may even be of infinite dimension (the gaussian kernel, for example).

Because kernel functions represent dot products in higher dimensional spaces, they allow us to map classification problems – implicitly – into a more separable space. Amazingly, we are able to both solve the problem and express the decision boundary in terms of K and the unmapped training set points, *without ever invoking the feature map itself*. Thus the name *the kernel trick*. Kernel logistic regression does precisely this. For example, if we consider the 3D data set represented here:



we will discover that a gaussian kernel is a much better performer than say, the linear kernel. It goes without saying that even though in this simple case we could create an explicit feature map ourselves, based on the shape of the data, for larger problems that is more difficult to do.

Common Kernels

Using no kernel is the same as implicitly using the linear kernel, which is the dot product in \mathbb{R}^n . Other popular kernels are:

1. Radial Basis Function Kernel (RBF): $K(x, x') = e^{-\frac{|x-x'|^2}{2\sigma^2}}$
2. Polynomial Kernel: $K(x, x') = (x^T x' + c)^d$
3. Sigmoid Kernel: $K(x, x') = \tanh(x^T x' + c)$

Gradient Descent for Kernel Logistic Regression

For standard logistic regression, recall that for input matrix X and labels y , we aim to minimize the

log-likelihood of the data, with regularization, given our weight vector w :

$$\min_w \frac{1}{n} \left[\sum_{i|y_i=1}^n \log \left(\frac{1}{1 + e^{-y_i \langle w, x_i \rangle}} \right) + \sum_{i|y_i=0}^n \log \left(1 - \frac{1}{1 + e^{-y_i \langle w, x_i \rangle}} \right) \right] + \lambda \|w\|^2,$$

which can be done via gradient descent:

$$w_0 = 0, \quad w_{t+1}^j = w_t^j - \gamma \left(\sum_i (y_i - \frac{1}{1 + e^{y_i \langle w, x_i \rangle}}) x_i^j + 2\lambda w_t^j \right)$$

For kernel logistic regression, the gradient descent iteration becomes:

$$c \leftarrow c + \gamma \left(y - \frac{1}{1 + e^{\hat{K}c}} + 2\lambda c \right),$$

where \hat{K} is the n by n kernel matrix consisting of the kernel function applied to each pair of training set points.

Conveniently, this does not depend on feature map Φ .

Using This Package

This package offers 3 types of gradient descent for solving kernel logistic regression: batch, stochastic, and mini-batch gradient descent. Let's briefly discuss the pros and cons of each.

Stochastic Gradient Descent

Stochastic gradient descent updates the weight vector by choosing a random example from the data set and calculating the error. The main advantage of this approach is two-fold: 1) it can be used in online settings, where data is acquired or revealed piece by piece, and 2) for non-convex loss functions, it can avoid local minima (not a worry with logistic regression). Drawbacks include a longer run time due to such frequent updates of the model, and a noisier gradient signal resulting in a higher variance of the model parameters during training.

For kernel logistic regression, the stochastic gradient descent iteration is given by:

$$c_i \leftarrow c_i + \gamma \left(y_i - \frac{1}{1 + e^{(\hat{K}c)_i}} + 2\lambda c_i \right)$$

The vector $\hat{K}c$ can be cleverly updated in linear time without recomputing the product each time.

Batch Gradient Descent

Batch gradient descent only updates the model after having seen all of the data points. In this way, it is usually more computationally efficient and more stable than its random cousin, but it may converge on a local minimum, and it often relies on the entire matrix being stored in memory. To boot, training time for batch descent can be very slow on large datasets.

Mini-batch Gradient Descent

Mini-batch gradient descent, a growing favorite of many, attempts to capture the best of both worlds. It randomly selects a batch-size number of points to update the model. The batch size may slide between 1 (stochastic) and n (batch), but usually a good choice is on the order of 30. It tries not to converge too slowly like batch, while simultaneously avoid the noise associated with a purely stochastic gradient descent. As a drawback, the batch size parameter may need to be tuned during validation.

For example usage of this package, please see `example.ipynb`.