Regression

Piyush Patil

March 10, 2017

In contrast to classification, regression problems are about learning the relationship between continuous variables. More specifically, given input vectors we want to predict numerical quantities that vary depending on the inputs; that is, we try to fit curves to data, accounting for random noise in the data. Like classification problems, regression will require us to select a loss function, to serve as a metric for how far our prediction for a data point is from its label or true value, and a cost function, which amalgamates the loss functions (usually a simple average) often with regularization terms. Most importantly, we choose a predictor function to model the data, which is really choosing a family of functions we think represent the data (i.e. linear, polynomial, logistic, etc.) and use techniques to find the parameters which give us the most likely specific curve to model the data.

Common loss functions include squared error, absolute error, and cross-entropy. Common cost functions are usually simply mean loss, with certain extensions such as weighting the loss at certain points more than at others (e.g. weighing outliers less) or by adding l_1 or l_2 regularization. One other option is simply the maximum loss.

1 Least squares

Least squares regression is one of the most common models in use. In this scheme, we use a linear predictor function and squared loss. More concretely, we try to find

$$\min_{w,\beta} \left(\sum_{i=1}^{n} (y_i - f(x_i))^2 \right) \text{ where } f(x) = w^\intercal x + \beta$$

In practice, we can make this more concise by shifting to a matrix representation of the data $\{(x_i, y_i), 1 \leq i \leq n\}$ by constructing a matrix X whose i^{th} row is x_i, and a vector y whose i^{th} entry is y_i. We can also add a column of 1's to the end of X put the shifting parameter β as the $(n+1)^{\text{th}}$ entry of w, to allow us to tune β as well (otherwise, we would be restricted to linear functions that pass through the origin). With our modified X and y, the problem becomes

$$\min_{w \in \mathbb{R}^{n+1}} |Xw - y|^2$$

which we can solve with some matrix calculus. Setting $E(w) := |Xw - y|^2$, we compute the gradient and find the minima:

$$\begin{aligned} E(w) = |Xw - y|^2 = (Xw - y)^\intercal (Xw - y) = w^\intercal X^\intercal X w - w^\intercal X^\intercal y - y^\intercal X w + y^\intercal y \\ = w^\intercal X^\intercal X w - 2 y^\intercal X w + y^\intercal y \text{ since } w^\intercal X^\intercal y \in \mathbb{R} \rightarrow w^\intercal X^\intercal y = y^\intercal X w \end{aligned}$$

$$\rightarrow \frac{\partial E}{\partial w} = 2X^\intercal X w - 2X^\intercal y \rightarrow \frac{\partial E}{\partial w} = 0 \text{ at } w = (X^\intercal X)^{-1} X^\intercal y$$

If $X^\intercal X$ is not invertible, then obviously the above solution doesn't exist. In this case, the problem is underconstrained, and infinitely many solutions exist.

Geometrically, we can interpret the above solution as follows. y is a vector in $\mathbb{R}^n$, and the function $f(w) = Xw$ has range given by the span of the columns of X, which is to say, the set of potential weightings of the features. Since $X \in \mathbb{R}^{n \times d}$, if $d < n$, which is almost always the case, then the range of f is a linear subspace of $\mathbb{R}^n$; specifically, it's constrained to a $(d+1)$-dimensional hyperplane. The problem of finding w which minimizes $|Xw - y|^2$ can then be interpreted as finding the point on that $(d+1)$-dimensional hyperplane which is closest to y in $\mathbb{R}^n$. If the assumption that the data is linear is indeed correct, and there's no random noise in the data, then y will be on this hyperplane and we'll be able to find it exactly; the more noise there is in the data, or the weaker our assumption of linearity is, the further y is from the hyperplane. Note that the nature of squared error loss functions is very sensitive to outliers.

2 Logistic Regression

Logistic regression is simply regression with a logistic predictor function and a cross-entropy (a.k.a. log-loss) cost function. Despite its name, logistic regression is a binary classification (though it can be extended to multiple classes) technique, in which we perform regression to approximate not a dependent variable but rather the *probability distribution* of the classes. Thus, the data labels y_i can be probabilities from the range $[0, 1]$, though in most practical cases they're simply binary classification labels $y_i \in \{0, 1\}$.
The logistic function

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

is ideal for estimating classification probabilities due to its smooth approximation of a step function.

$$\lim_{x \to -\infty} s(x) = 0 \text{ and } \lim_{x \to \infty} s(x) = 1$$

In particular, what we'll do is parameterize the logistic function by computing, for input feature vector x, the prediction $p = s(w^\intercal x + b)$ where w is our weight vector and b is the bias; these are the quantities we want to learn. Unlike some of the other classification models, the output p isn't an estimate of what class x falls into, but rather an estimate of the probability that x falls into one of the two classes. In other words, we're making the assumption that

$$\Pr(y = 1|x) \approx s(w^\intercal x + b)$$

for appropriate w. Logistic regression therefore consists of two parts. First, as with support vector machines from linear classification, we assume that the input space is linearly separable, and hence try to learn a linear decision boundary to classify inputs; as usual, we use w to parameterize the decision boundary, and interpret it as being the normal vector defining the separating hyperplane. The quantity $w^\intercal x$ is positive if and only if x lies above the hyperplane through the origin determined by normal vector w, so we add in the bias term to allow for hyperplanes that don't pass through the origin. Second, we take the quantity $w^\intercal x + b$, which is some notion of *evidence* for whether x belongs to the class $y = 1$, and convert it to a probability by passing it through the logistic function.
This raises the question of why the logistic function is a good mapping to use to map evidence based on a linear decision boundary to a probability. To understand this, it's useful to consider the inverse of the logistic function - the *logit* function, given by

$$s^{-1}(p) = \log\left(\frac{p}{1 - p}\right)$$

This function has range $(0, 1)$, and we if we interpret p as a probability, then the above quantity is precisely the *log-odds*. Odds are an alternate probabilistic framework wherein rather than modeling the probability of the event, we look at the ratio of the probability of success against the probability of failure. Odds are therefore a metric that encode the relative likelihood of an event, rather than the raw, absolute probability. Putting the odds through a logarithm is a practical trick that makes the quantity easier to analyze, since additive changes in the log-odds correspond to changes in the odds ratio, which is a measure of the relative likelihood of an event occurring under different conditions. The motivation for using the logistic function, then, is that we interpret the evidence $w^\intercal x + b$ as a log-odds, and map it to a probability with s. This means what our model is learning when it tunes w and b is a way to compute the log-odds of the event $y = 1$ given x.
Geometrically, what we're doing here is assuming that each data point x is not a featurization of the true input, but rather the result of mapping the true input to its log-odds for the event that the input falls into the class $y = 1$. Given this assumption, we then try to fit a logistic curve to the data, because given the logistic function of best fit we can most optimally convert the log-odds to absolute probabilities, obtaining the probability that the true input is in the class $y = 1$.
We use the cross-entropy loss function (the intuition for this loss function is given in the below sections), given by

$$J(w) = \sum_{i=1}^{n} \left(y_i \log(s(w^\intercal x_i)) + (1 - y_i) \log(1 - s(w^\intercal x_i))\right)$$

which happens to be convex, allowing us to perform logistic regression with gradient descent and guarantee convergence to a global minimum. Moreover, the logistic function has the convenient property

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

We can compute the optimal weight w with gradient descent. To do so, we now find the derivative of the log-loss function with respect to w, which can be used in the gradient descent algorithm. It follows that for data matrix X, label vector y,

and paramter w, as described in the previous section, we have

$$\begin{aligned}\frac{\partial J}{\partial w} &= \sum_{i=1}^{n}\left(\frac{y_i}{s_i}\frac{\partial s_i}{\partial w} - \frac{1-y_i}{1-s_i}\frac{\partial s_i}{\partial w}\right) \text{ where } s_i = s(w^\intercal x_i) \\ &= \sum_{i=1}^{n}\left(\frac{y_i}{s_i} - \frac{1-y_i}{1-s_i}\right) s_i \cdot (1-s_i)x_i \\ &= \sum_{i=1}^{n}(y_i - s_i)x_i = X^\intercal(y-s)\end{aligned}$$

where s above is the vector of s_i's.

3 Newton's Method

This is an iterative optimization method for finding function minima that often converges much faster than gradient descent. The idea is simply to truncate the Taylor series of the loss function J which we're optimizing at the second-order term, thereby using a linear approximation. Assuming J is (infinitely) differentiable, by Taylor's theorem

$$\frac{\partial J}{\partial w}(w) = \frac{\partial J}{\partial w}(w_0) + \frac{\partial^2 J}{\partial w^2}(w_0)\cdot(w-w_0) + O(|w-w_0|^2)$$

Ignoring the error term at the end and setting the derivative to zero,

$$w = w_0 - \left(\frac{\partial^2 J}{\partial w^2}(w_0)\right)^{-1}\frac{\partial J}{\partial w}(w_0)$$

where w_0 is just some starting point and the second derivative is the Hessian matrix. We can iteratively apply the above equation with successively closer starting points w_0 by using the approximation at one iteration as the starting point for the next, and quickly converge to the optimum. Thus, the update rule is given by

$$w_{k+1} = w_k + \left(\frac{\partial^2 J}{\partial w^2}(w_k)\right)^{-1}\frac{\partial J}{\partial w}(w_k)$$

Note that because we truncated at the quadratic term, if J is quadratic then the method will converge in a single iteration, as the solution is exact. One thing to keep in mind here is that Newton's method doesn't differentiate between maxima, minima, or saddle points; it merely finds the closest extremum to w_0, and thus is heavily dependent on the choice of starting point. Moreover, computing the Hessian matrix every iteration can get expensive in high dimensions.

4 Least Squares Polynomial Regression

This is essentially the same as the least squares regression method described above, but with a general polynomial (with a given maximum degree) predictor function; thus, linear least squares is a special case of this method. The problem to solve here is

$$\min_a\left(\sum_{i=1}^{n}(f(x_i)-y_i)\right) \text{ where } f(x) = \sum_{j=0}^{p} a_j x^j$$

where p is the maximum degree of the polynomial we're using. Solving this can be difficult, but another way to perform polynomial regression is to adapt least squares regression by first putting the feature vectors through the appropriate transformation. Specifically, we simply treat x_i^d as a separate variable, thereby mapping each $x_i \in \mathbb{R}^d$ to a higher dimensional vector that we can perform linear regression on.

5 Bias-Variance Decomposition

In this section we justify the use of the above loss functions, and go over the tradeoff between bias and variance of a model. Let's start off with a high-level discussion on machine learning problems and their solutions. Typically, when applying machine learning to a problem, we first look at the data we have available, come up with a model for the data, state an optimization problem whose solution will allow us to build the specific model, and finally choose an optimization algorithm to solve the optimization problem. So far, in both classification and regression, we've been looking at the last two stages of this process - the optimization problem and algorithms for solving it. In this section, we'll be looking at and justifying

different machine learning models.
In general, we have feature vectors x_i representing our data, which we model as coming from some unknown underlying probability distribution, say D. We model the labels, y_i as some unknown, continuous function f of the data, with some random noise thrown in:

$$y_i = f(x_i) + \epsilon_i \text{ where } \epsilon_i \sim D'$$

where D' is the probability distribution of the random noise ϵ, which we also assume has mean at zero. This reflects the assumption that in expectation, we should have $\mathbb{E}(y_i) = \mathbb{E}(f(x_i))$ since after all ϵ is merely random noise which should average out in the long run. The goal of regression is to find a function h which approximates f. Note that our assumptions that

1. f is smooth.
2. ϵ_i doesn't depend on x_i

are quite strong, and don't always hold in practice.

Justification for least squares: Let's justify and motivate our use of the squared error loss function, $L = \sum_i (y_i - f(x_i))^2$. This function has some nice properties, such as being convex and representing an intuitive notion of distance, but we can more rigorously motivate its use. It turns out that when the random noise ϵ_i in the mapping from feature vectors to their labels follows a Gaussian distribution, then according to maximum likelihood estimation, approximating f is best done by minimizing the squared error loss function. To see this, notice that if $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ then $y_i \sim \mathcal{N}(f(x_i), \sigma^2)$. The log-likelihood for the Normal distribution is

$$\log(\mathcal{L}((x_i, y_i)|\epsilon_i)) = \log(\Pr(y_i)) = -\frac{((y_i - f(x_i))^2}{2\sigma^2} - \log(\sigma\sqrt{2\pi})$$

$$\rightarrow \log\left(\prod_i \Pr(y_i)\right) = -\frac{1}{2\sigma^2}\sum_i (y_i - f(x_i))^2 - C$$

where C is some constant. Thus, according to MLE, we want to maximize the above function for our approximation of f, which is equivalent to minimizing the squared loss function

$$L = \sum_i (y_i - f(x_i))^2$$

Justification for cross-entropy: Let's now justify and motivate the use of the cross-entropy loss function, given by

$$L = -\sum_i \left(y_i \log(f(x_i)) + (1 - y_i)\log(1 - f(x_i))\right)$$

This loss function is optimal in classification problems when we're modeling the probability that a feature vector belongs to a certain class. We focus on the special case of binary classification, which extends to multi-class classification easily. Let $h(x)$ be the probability that x is in class $y = 1$ (as opposed to $y = 0$). Then

$$\Pr(y_i = 1|x_i) = h(x_i) \text{ and } \Pr(y_i = 0|x_i) = 1 - h(x_i)$$

$$\rightarrow \Pr(y_i|x_i) = \begin{cases} h(x_i), \text{ if } y = 1 \\ 1 - h(x_i), \text{ if } y = 0 \end{cases} = (h(x_i))^{y_i} \cdot (1 - h(x_i))^{1-y_i}$$

Under maximum likelihood for parameter w on which h depends, we want to maximize

$$\log\left(\prod_i \Pr(y_i)\right) = \log\left(\prod_i (h(x_i))^{y_i} \cdot (1 - h(x_i))^{1-y_i}\right) = \sum_i \left(y_i \log(h(x_i)) + (1 - y_i)\log(1 - h(x_i))\right)$$

The cross entropy loss function is simply the negative of the above function, and so maximizing the above function, as MLE tells us to do, is equivalent to minimizing the cross-entropy function.

6 Regularized Regression

In this section we'll look at applying some commonly used regularization terms, which prevent overfitting by enforcing simpler models, to regression.

6.1 Ridge Regression

This is a simple but effective use of regularization. Ridge regularization is essentially simply l^2 regularized linear least squares, which means our loss function is simply

$$L(w) = |Xw - y|^2 + \lambda |w'|^2$$

where, as usual, X is our data matrix for $\{(x_i, y_i), 1 \leq i \leq n\}$, whose rows are our feature vectors x_i, y is the vector of labels y_i, and w is a vector of weights with a final translation term appended at the end, so that $(Xw)_i = x_i^\intercal w + \beta$. As usual, we want to minimize the squared error between our predictions Xw and the labels y, but we include the regularization term to encourage *shrinkage*; in other words, to encourage small weights, representing our preference for simpler models over complex ones. Above, w' is defined to be identical to w but with the last entry, representing β, set to zero. This is because we don't want to regularize the translation term, which has no influence on the complexity of our model and simply slides it up and down in space.

The solution the the new problem is a slightly modified form of the original solution to unregularized least squares:

$$w = (X^\intercal X + \lambda I')^{-1} X^\intercal y$$

where I is the identity matrix modified so that the bottom right entry is 0 instead of 1 (this is to avoid penalizing the translation term β). The effect of ridge regression is, geometrically, to accentuate the convexity of our loss function by adding a "ridge" through the minima; whereas an unregularized least squares loss function, which is still parabolic in nature, might have multiple or even infinitely many minima, regularization adds an additional factor which forces the curve to conform to a slight ridge that reduces the number of minima, and does so inversely proportional to the size of the w to which the minima corresponds so that smaller w's minima are maintained with the rest forced up along the ridge. Mathematically, we can see in the above equation that we're adding a "ridge", proportional to λ, to the main diagonal of the matrix $X^\intercal X$, guaranteeing that the matrix is invertible and therefore that we obtain a unique minimum. As with all regularization, from the model's perspective, we're reducing the variance. In fact, as $\lambda \to \infty$, the variance approaches zero, though the bias may (probably will) increase.

Bayesian justification: Similar to how we justified using a least squares cost function as being optimal according to MLE if we assume a Guassian distribution on our error term, we can justify linear regression from a Bayesian perspective as well. We'll then use the derived equations to insert an additional assumption that will illustrate where and why the l^2 norm regularization term comes up. To derive the basic ordinary least squares equations, we make two assumptions:

1. A linear model of the data is, in fact, optimal. This means that from a bias-variance perspective, we have

   $$y_i = w^\intercal x_i + \epsilon$$

   where ϵ is some random error term. Recall that we modeled a general regression problem as trying to approximate f where $y = f(x) + \epsilon$ for some smooth f; all this assumption is stating is that f is linear, so that we can proceed with the linear model above.

2. As we did when justifying a least squares cost function, we assume that ϵ is Gaussian noise, i.e. $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

With these assumptions, it follows that, for parameters w, σ,

$$\Pr(y_i|x_i) = \mathcal{N}(w^\intercal x_i, \sigma^2) \text{ and } \Pr(y|X) = \prod_i \Pr(y_i|x_i)$$

This just means that for a given feature vector and optimal w, the label follows a Gaussian distribution centered exactly at what we predict (i.e. $w^\intercal x_i$, since we did, after all, assume that a linear model was optimal) and with variance dependent on the random noise. Applying maximum likelihood to find the optimal value of w above, we maximize the log-likelihood, which is given by

$$\log(\Pr(y|x, w, \sigma^2)) = \sum_i \log(\mathcal{N}(y_i|x_i, w, \sigma^2)) = \sum_i \log\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{(y_i - w^\intercal x_i)^2}{2\sigma^2}\right)\right) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_i (y_i - w^\intercal x_i)^2$$

which has minimum at exactly what we expect: $w = (X^\intercal X)^{-1} X^\intercal y$. Note that we obtained this from the assumption that

$$\Pr(y|X) \propto \exp\left(\frac{1}{2\sigma^2}\sum_i (y_i - w^\intercal x_i)^2\right)$$

The key here is that if we make a third assumption on the distribution of the parameter w as well that

3. $\Pr(w) \sim \mathcal{N}(0, 1)$

in addition to the first two, then we have

$$\Pr(y|X) = \Pr(y|X, w)\Pr(w) \propto \exp\left(\frac{1}{2\sigma^2}\sum_i (y_i - w^\intercal x_i)^2\right) \cdot \exp\left(|w|^2\right) = \exp\left(-\frac{1}{2\sigma^2}\sum_i (y_i - w^\intercal x_i)^2 - |w|^2\right)$$

which reduces the optimization problem composed of exactly of what we obtained earlier (minimizing the sum of squared errors) plus the l^2 norm of w. We can throw the scaling factor λ into the distribution of w to tune it. Then, the problem has the solution we expect from ridge regression: $w = (X^\intercal + \lambda I)^{-1} X^\intercal y$, showing that if we make the three assumptions above, ridge regression is indeed optimal.

6.2 Lasso Regression

LASSO (least absolute shrinkage and selection operator) regression is very similar to ridge regression; it's simply l^1 regularized linear least squares regression. In other words, we're minimizing the loss function

$$L(w) = |Xw - y|^2 + \lambda |w'|_1$$

where, as before X is the data matrix, y is the vector of labels, and w is our weight parameter. Recall that

$$|w'|_1 = \sum_{i=1}^{d} |w_i| \text{ where } w \in \mathbb{R}^{d+1}$$

The key difference between ridge regression and lasso regression, which both emphasize smaller weights, is that lasso regression will actually force some of the weights to vanish entirely (i.e. be set to zero), forcing the model to identify which features are useful and which are redundant or don't correlate well with the labels. This provides *variable selection*, telling us which features are valuable, which aren't, and quantifiably by how much, which improves the interpretability of the model.

Geometric explanation: Geometrically, this is because while both ridge and lasso regression minimize the same objective function, lasso regression imposes a constraint on the weights that forces them to lie on a cross-polytope (a hypercube centered at the origin and rotated so that it's vertices lie on the coordinate axes) whereas the constraint imposed by ridge regression forces the weights to lie on a hypersphere centered at the origin. This is because unit balls (or neighborhoods) given by the level curves of the l^1 and l^1 metrics. Hyperspheres are rotationally invariant, which means to optimize the objective function we only need to minimize a point on the objective surface which touches any point on the sphere. Since the cross-polytope, however, is not rotationally invariant, minimized points on the objective surface that lie on the cross-polytope are more likely to lie near or on a vertex, which forces some of the parameters to be set to zero.

Bayesian justification: From a Bayesian perspective, lasso regression arises where, in addition to the two assumptions we made in ridge regression (namely, that a linear model is indeed optimal and that the model's error is Gaussian noise), we impose a Laplace prior distribution on the weights (as opposed to a Guassian prior as we did in ridge regression).