Linear Regression (Least Squares)

Linear regression assumes the target is a linear function of the input features. Concretely, we choose hypotheses of the form $h(x) = w_0 + w_1x$ (or in general $h(x) = w^Tx$ if x includes a constant 1 for the bias). We fit this model by minimizing the mean squared error (MSE) on the training set:

$$L_S(w) = \frac{1}{m}\sum_{i=1}^{m}(w^\top x_i - y_i)^2.$$

This is called **least-squares regression** [1] [2]. Geometrically, the learned line (or hyperplane in higher dimensions) is the one that minimizes the average squared vertical distance to the data points. By setting the gradient of the loss to zero, one obtains the normal equations, giving the closed-form solution

$$w_S = (Z^T Z)^{-1} Z^T y,$$

where *Z* is the design matrix of all input vectors augmented with a 1 for the bias term [3]. In other words, the **least-squares solution** has a simple matrix formula that can be computed directly from the data.

Importantly, the model includes an explicit *bias* or intercept term. Writing each input as *z_i = (x_i,1)* allows the model w^T*z_i* = w_1*x_i* + w_0; here w_0 is the bias that shifts the fitted line up or down. Including a bias lets the hyperplane fit data that do not pass through the origin, improving flexibility [4]. In summary, linear regression fits a hyperplane by minimizing squared error; its solution is known as the least-squares solution, and it can be computed in closed form from the feature matrix (including the bias column) [3] [5].

Metric Spaces and Norms

To reason about distances and regularization, we view our vector spaces as **metric spaces** with norms. A norm provides a distance function $|u|*p = (\sum*$ on vectors. For different values of }^d |u_j|^p)^{1/ p*p*, this gives different behaviors: *p=2* yields the Euclidean norm (standard L2 distance), *p=1* gives the Manhattan (L1) distance, and as *p→∞* one obtains the max-norm (L∞) [6]. These norms satisfy the usual metric properties (positivity, symmetry, triangle inequality, etc.), so we can measure similarity or size of vectors in these spaces. In machine learning, choosing an L1 versus L2 norm has geometric consequences: for example, the unit-ball of L2 is round, while L1's unit-ball is diamond-shaped (with corners on the axes). This will later explain why L1 regularization can drive some weights exactly to zero (sparsity) whereas L2 tends to shrink all weights more uniformly [6].

Regularization and Ridge Regression (L2)

Fitting a model with no constraints can lead to overfitting (memorizing noise). To prevent this, we use **regularization** to control model complexity. A classic idea is **structured risk minimization**, where we add a penalty term to the loss. For example, we can solve

$$\min_w \frac{1}{m}\sum_{i=1}^{m}(w^\top x_i - y_i)^2 + \lambda\|w\|_p^p,$$

where $\lambda>0$ is a **regularization coefficient** and $|w|_p^p$ is a penalty on the weights [7]. This is equivalent to constraining $|w|_p\le\eta$ for some bound. A key case is $p=2$, giving the **ridge regression** objective

$$\frac{1}{m}\sum_{i}(w^\top x_i - y_i)^2 + \lambda\|w\|_2^2.$$

In vector form the loss is

$$L_S(w) = \frac{1}{m}(Zw-y)^\top(Zw-y) + \lambda\, w^\top w,$$

where Z is the design matrix [8]. Setting its gradient to zero yields the ridge solution

$$w_S = \left(\tfrac{1}{m}Z^TZ + \lambda I\right)^{-1} Z^T y.$$

Compared to ordinary least squares, ridge regression adds λI to the matrix being inverted [9]. This has the effect of *shrinking* weights toward zero, reducing variance. In statistics, methods like ridge that "pull" (shrink) parameters in are called **parameter shrinkage** methods [10], and the $ \lambda|w|_2^2$ penalty is sometimes called *weight decay*. Ridge regression often improves generalization by balancing fit (MSE) and complexity (ℓ_2 norm of *w*) [9] [10].

Lasso Regression (L1 Regularization)

Another regularization approach is to use the L1 norm of the weights. Setting $p=1$ leads to the **LASSO** (Least Absolute Shrinkage and Selection Operator) loss [11]:

$$L_S(w) = \frac{1}{m}\sum_{i=1}^{m}(w^\top x_i - y_i)^2 + \lambda\sum_{j=0}^{D}|w_j|.$$

Geometrically, the L1 penalty encourages many weights to become exactly zero (parameter sparsity). Unlike ridge, there is no closed-form solution for the Lasso: the L1 term makes the objective non-differentiable at zero, so we cannot simply set a gradient to zero [12]. In practice, Lasso is solved by iterative optimization (e.g. coordinate descent or gradient methods). Its main virtue is sparsity: as observed in the notebook, "Lasso sparsifies the parameters more than ridge," meaning it tends to drop less important features entirely [13]. In summary, L1 regularization (Lasso) uses the same squared loss as before but adds an absolute-value penalty, leading to a sparse linear predictor with no simple algebraic solution [11] [12].

Gradient Descent Optimization

When a loss has no analytic minimizer (as with Lasso) or when closed-form is impractical, we use iterative optimization like **gradient descent**. Starting from an initial weight vector w_0, gradient descent updates iteratively via

$$w_{t+1} = w_t - \alpha\nabla_w L_S(w)\big|_{w=w_t},$$

where $\alpha>0$ is the learning rate [14]. Intuitively, at each step we move a small amount in the direction opposite the loss gradient, which locally points uphill; stepping against it goes downhill. The choice of α is crucial: too large a learning rate can overshoot the minimum (oscillate), while too small a rate makes convergence very slow [15]. In modern practice, tools like PyTorch or TensorFlow compute gradients automatically, and they iteratively apply such updates. Gradient descent applies broadly across models; here it is illustrated with Lasso regression, which is trained by repeatedly computing gradients of the MSE+L1 loss and stepping until convergence [14] [16].

Learning Curves

A **learning curve** plots model performance (e.g. error) over training progress (iterations or epochs). For example, one can plot training and test error versus iteration count to see how the model improves. The notebook describes that “learning curves give plenty of information about the model behavior” and are useful for debugging [17]. A typical learning curve may show error dropping on the training set as gradient descent proceeds, while the test error may decrease then plateau or increase (indicating overfitting). By inspecting such curves, one can detect issues like a model not learning at all (flat curve) or diverging (error grows), and adjust learning rate, regularization, or model complexity accordingly [17]. In short, learning curves visualize the evolution of loss (or accuracy) during training and help diagnose the training process.

Sources: Concepts and formulas are drawn directly from the lecture notebook, including the definitions of linear models, least-squares solutions [1] [3], Lp norms [6], regularized loss functions [18], the ridge solution [9], Lasso loss and gradient descent updates [11] [14], and the explanation of learning curves [17].

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] 02_Linear_Predictors.ipynb
file://file-FfSsJ6ZCNyADF8tBfoMX1T