Useful Matrix Algebra for Regression Models

Francisco Pereira

February 9, 2023

1 Notation

X is the $n \times d$ data matrix, with n examples and d dimensions. X can be either wide ($d \gg n$) or narrow ($n \gg d$); whenever this makes a difference, it will be noted. Y is the $n \times t$ regression target matrix, with n examples and t targets. A further assumption is that the examples are divided into g groups, and that we can specify the group specific data and target matrices X_g and Y_g; hence, $X = [X_1; \ldots \ X_g]$. I will abuse MATLAB notation and use "," and ";" to specify horizontal and vertical matrix concatenation, respectively. A final assumption is that every dimension and target have been centered, i.e. they have mean 0. This simplifies many of the formulas, and I will note the places where this is critical. I_d is the identity matrix with d dimensions.

2 Singular Value Decomposition

2.1 Definition

The singular value decomposition (SVD) of X is

$$X = USV'$$

where U is $n \times k$, S is $k \times k$, and V is $d \times k$. The matrices U and V are called the left and right singular vectors, and S are the singular values.

The matrices have various properties that will be used in the rest of the document:

- U and V are orthonormal: their columns have norm 1, and are orthogonal to each other, so $U'U = I_k$ and $V'V = I_d$ are both matrices with ones in the diagonal, and zeros elsewhere.
- S is a diagonal matrix, with different values in each entry of the diagonal.
- As orthogonal matrices, U and V can be viewed as *bases* for the spaces spanned by columns or rows of X, respectively, and where coordinates for X would be SV' or US.
- k can only be as large as $\texttt{min}(n, d)$; if X is of rank lower than that, which can happen if columns or rows are linearly dependent, it can be smaller.

2.2 Relationship with the original matrix

SVD provides the best approximation of the matrix X at any specified rank. I.e. one can take the full SVD and expand it to show that it reconstructs X as a summation of k rank 1 matrices

$$X = U_{:,1}S_{1,1}V'_{:,1} + \ldots + U_{:,k}S_{k,k}V'_{:,k}$$

Each of these matrices explains some of the variance in X, and the dimensions of the SVD are sorted from those that explain the most to those that explain the least. The rank of the approximation is determined by the number of matrices that are added.

And how to determine the rank to use? If we take the diagonal of the matrix S, $\mathtt{s} = \mathtt{diag}(S)$, then the percentage of variance accounted for by dimension i is $\frac{s_i^2}{\sum_{j=1}^{k} s_j^2}$. Adding up the variance for dimensions $1-i$ will give you the total variance explained by the rank i approximation. This is useful in data compression (best approximation), denoising (many dimensions with little variance are capturing noise), or in matrix completion (if there are missing entries, the SVD reconstruction can be a good multivariate imputation method, subject to certain conditions).

2.3 Dimensionality reduction

The most common use of SVD, besides the ones above, is for dimensionality reduction. As mentioned earlier, if we consider V' as a k vector basis for the examples in the rows of X, it means that the examples can be represented as k-dimensional vectors in that space. The corresponding coordinates are simply US. In addition to reducing dimensionality, this also ensures that the variables in the low-dimensional space are uncorrelated, which make them suitable for various applications (e.g. visualization, preprocessing for other methods that assume uncorrelated variables).

The other nice property of the SVD is that the matrix V can be used *both* to convert X (or other examples) in the d-dimensional space into the k-dimensional space, as

$$\begin{aligned} X &= USV' \\ XV &= USV'V \\ XV &= US \text{ (as } V'V = I_k) \end{aligned}$$

and also convert any examples in the low-dimensional space back into the d-dimensional space, by multiplying them by V' (as $X = (US)V'$).

2.4 Relationship with Principal Component Analysis

The covariance matrix for a *centered* dataset X is $C = \frac{1}{n}X'X$. Principal Component Analysis is a spectral decomposition of C, i.e.

$$C = E\Lambda E'$$

where E is a $d \times d$ matrix, and Λ is a diagonal matrix. The E is an orthonormal matrix (column vectors are norm 1, and orthogonal to each other), where columns are called *eigenvectors*. The diagonal of Λ, $\lambda = \mathtt{diag}(\Lambda)$ contains the corresponding *eigenvalues*. The matrix E acts as a basis for reconstructing the covariance matrix E. The fraction of variance explained by eigenvector i is $\frac{\lambda_i}{\sum_{j=1}^{d} \lambda_j}$.

So what is the relationship between PCA and SVD? If $X = USV'$, then C can be expressed in terms of the SVD matrices, as follows

$$\begin{aligned} C = \frac{1}{n}X'X &= \frac{1}{n}VS'U'USV' \\ &= V\frac{1}{n}S'SV' \text{ (as } U'U = I) \\ &= V\frac{1}{n}S^2V' \text{ (as } S = S') \end{aligned}$$

so the eigenvector matrix E is the same as the right singular vector matrix V (with some caveats if X is not full rank). This also explains why we have to use S^2 when calculating the percentage of variance explained by each singular vector.

When would we not use PCA, then? If X is wide ($d \gg n$), then doing a spectral decomposition of the covariance matrix C would be problematic for two different reasons: not enough data points to properly estimate C, and doing the decomposition of a potentially very large $d \times d$ matrix. Which brings us to...

2.5 Efficiency considerations in estimating the SVD

If one of the dimensions of X is much larger than the other, there is a possibility that some SVD implementations will fail through memory usage, or simply take too long. However, there are simple, efficient ways of estimating the SVD that work for any implementation. If $X = USV'$, then we have:

- XX' is a $n \times n$ matrix, and $XX' = USV'VS'U' = US^2U'$
- $X'X$ is a $d \times d$ matrix, and $X'X = VS'U'USV' = VS^2V'$

i.e. as U and V are orthonormal, and S is diagonal, the SVD gives us a spectral decomposition for both XX' and $X'X$. So, to compute it efficiently via whatever dimension is smaller:

- if $d \gg n$, then SVD of XX' gives us U and S, and $V = X'US^{-1}$
- if $n \gg d$, then SVD of $X'X$ gives us V and S, and $U = XVS^{-1}$

3 Ridge Regression

3.1 Matrix algebra for (ridge) regression

For a multivariate (predict matrix Y with t targets) multiple (from matrix X with d variables) linear regression problem, the goal is to find a weight matrix W such that

$$Y \sim XW$$

where column $W_{:,j}$ are the weights for target j. The normal equations can be expressed in matrix algebra as

$$W = (X'X)^{-1}X'Y$$

For wide matrices X this will be extremely numerically unstable, or fail, as the colinearity means that $X'X$ is not invertible. This could also happen for narrow matrices, if variables are extremely correlated. A ridge regression adds a small value λ to the diagonal of the $X'X$ matrix

$$W = (X'X + \lambda I_d)^{-1}X'Y$$

making the resulting matrix invertible and the problem solvable. This λ is equivalent to the regularization weight on the norm of each weight vector w (column of W), in an optimization formulation of the problem

$$\arg\min_{\mathbf{w}} \|X\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$$

Note that, if doing this through matrix algebra for multiple targets, this method applies the *same* regularization to every target, which may not be the appropriate (e.g. if many variables have information about one target, but only very few do about another). In that case, one should solve this separately for each target column $Y_{:,t}$.

3.2 Relationship between ridge regression and SVD

If we consider the SVD of X, $X = USV'$, and replace it in the normal equations

$$\begin{aligned} W &= (X'X)^{-1}X'Y \\ &= (VS'U'USV')^{-1}X'Y \\ &= (VS^2V')^{-1}X'Y \end{aligned}$$

they will work, as VS^2V' is invertible. Why is this? Because the SVD will not produce more singular vectors than the rank of X. In fact, if we keep only as many singular vectors as needed to capture a certain percentage of variance, the effect will be similar to increasing the λ penalty in the previous section.

A different way of using the SVD would be to replace X by a low-dimensional representation US, as we saw earlier. Given that the dimensions of that matrix are uncorrelated, we can use them in the original normal equations. The weight vector for each regression target would be k-dimensional, but it could be converted back to the d-dimensional space by multiplying by V'. (TODO: demonstration). As above, this is implicitly equivalent to setting a particular λ penalty in the previous section.

3.3 Making (cross-validated) regression more efficient

Cross-validation If we do cross-validation using G groups, in each fold we will be training a model using examples from G-1 groups, and testing on the remaining one. Without loss of generality, assume we are using groups $1, \ldots, G-1$ for training. The training set is $X_{train} = [X_1; \ldots; X_{G-1}]$ and the corresponding targets are $Y_{train} = [Y_1; \ldots; Y_{G-1}]$; the test set is $\mathbf{x}_{test} = X_G$ with targets $Y_{test} = Y_G$.

If using ridge regression, the weights in fold g will be

$$W_g = (X'_{train}X_{train} + \lambda I_d)^{-1} X'_{train}Y_{train}$$

and will then be applied to the test set $\mathbf{x}_{test}$ to generate predictions.

Optimization In each fold, we do two matrix multiplications, $X'_{train}X_{train}$ and $X'_{train}Y_{train}$. Given that the training sets mostly overlap across folds, many of the calculations are repeated. This computation may be accelerated by considering that

$$X'_{train}X_{train} = [X'_1, \ldots, X'_{g-1}][X_1; \ldots; X_{g-1}] = X'_1X_1 + \ldots + X'_{g-1}X_{g-1}$$

and, likewise

$$X'_{train}Y_{train} = [X'_1, \ldots, X'_{g-1}][Y_1; \ldots; Y_{g-1}] = X'_1Y_1 + \ldots + X'_{g-1}Y_{g-1}$$

Hence, we can do the whole cross-validation in two stages

1. before the cross-validation loop, compute:
   - terms X'_gX_g and X'_gY_g for each group g
   - sums $A = \sum_g X'_gX_g$ and $B = \sum_g X'_gY_g$
2. in each fold g, compute:
   - $X'_{train}X_{train} = A - X'_gX_g$
   - $X'_{train}Y_{train} = B - X'_gY_g$

which means the whole process requires only G matrix multiplications and matrix-vector multiplications, and otherwise matrix additions/subtractions. This can, potentially, be combined with other techniques for making the matrix inversion more computationally efficient, which will be discussed in the next section.

4 Kernel Ridge Regression

4.1 Method

The first basic idea motivating kernel ridge regression is that the ridge regression weights $\mathbf{w}$ for a given target are a linear combination $\mathbf{a}$ of the training examples X, i.e. $\mathbf{w} = X'\mathbf{a}$ (TODO: derivation using the Woodbury formula). I will use a single prediction target throughout this section for clarity. The second is the notion of a kernel matrix. For now, let's consider the simplest one, the linear kernel $K = XX'$ (also

known as the Gram matrix), where $k_{ij} = X_{i,:}X'_{j,:}$, the dot product between examples i and j. Intuitively, this gives us a (non-normalized) measure of similarity between the examples.

Then, the ridge regression optimization problem in terms of $\mathbf{w}$ and X:

$$\arg\min_{\mathbf{w}} \|X\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$$

can be transformed to an equivalent one in terms of $\mathbf{a}$ and K:

$$\begin{aligned}
\mathbf{a}^* &= \arg\min_{\mathbf{a}} \|XX'\mathbf{a} - \mathbf{y}\|_2^2 + \lambda\|X'\mathbf{a}\|_2^2 \\
&= \arg\min_{\mathbf{a}} \|K\mathbf{a} - \mathbf{y}\|_2^2 + \lambda\mathbf{a}'K\mathbf{a} \\
&= (K + \lambda I_n)^{-1}\mathbf{y} \text{ (analogously to ridge regression)}
\end{aligned}$$

What are the advantages of this transformation? When applying the learned weights to a test set $\mathbf{x}_{test}$ to generate a prediction, we have

$$\begin{aligned}
\mathbf{x}_{test}\mathbf{w} &= \mathbf{x}_{test}X'\mathbf{a} \\
&= K_{test}\mathbf{a}
\end{aligned}$$

and, therefore, the predictions for each test example (row of $\mathbf{x}_{test}$) are a function of how similar it is to the examples in the training set (the kernel matrix K_{test}). Furthermore, the contribution of each training set example to the prediction comes from $\mathbf{a} = (K + \lambda I_n)^{-1}\mathbf{y}$ (each row of $(K + \lambda I_n)^{-1}$ acts as a linear combination of values in $\mathbf{y}$). Overall, examples only appear in the formulas through kernel matrices, i.e. how similar they are to other examples, never by themselves.

4.2 Advantages of kernel ridge regression

Computational efficiency The kernel matrix K is square ($n\times$n) and symmetric. It is positive semi-definite, as it can have a rank less than n (e.g. if $n \gg d$); if so, some of the eigenvalues in a spectral decomposition may be 0, and the matrix will not be invertible. If we consider the SVD of $X = USV'$, then $K = USV'VS'U' = US^2U'$. Using this, the formula for the optimal example weights $\mathbf{a}$ can be re-written (TODO: derivation)

$$\mathbf{a} = (K + \lambda I_n)^{-1}\mathbf{y} = U(S^2 + \lambda I_n)^{-1}U'\mathbf{y}$$

If fitting the model for different values of the regularization parameter λ, this will be considerably more efficient, as it only requires inverting the diagonal matrix $(S^2 + \lambda I_n)$.

The "kernel trick" In all of the definitions in the previous section, both training and test examples only appeared as part of a kernel matrix. Apart from computational efficiency in the sense described above, this makes it possible to do all the ridge regression computations *as if* the examples were in other, higher-dimensional feature spaces. To see how, consider a function $\phi(\mathbf{x})$ mapping each example $\mathbf{x} = [x_1, \ldots, x_d]$ to a new example $\phi(\mathbf{x}) = [\phi_1(\mathbf{x}, \ldots, \phi_m(\mathbf{x})]$. For instance, in the case where $\mathbf{x}$ is two dimensional, we could have a new, three-dimensional feature space that considers each coordinate in isolation, and their product: $\phi([x_1, x_2]) = [x_1^2, x_2^2, \sqrt{2}x_1x_2]$. The reason for doing this might be that examples become separable (for a classifier) in the new feature space, e.g. the classic XOR example that is impossible for linear classifiers but is feasible once products of features are introduced. The product of examples $\phi(\mathbf{x_i})$ and $\phi(\mathbf{x_j})$ would be $K_{ij} = \phi(\mathbf{x_i})\phi(\mathbf{x_j})' = x_{i1}^2x_{j1}^2 + x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2}$.

However, there is another way of computing this, namely by using a kernel function:

$$\begin{aligned}
\phi(\mathbf{x_i})\phi(\mathbf{x_j})' &= x_{i1}^2x_{j1}^2 + x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} \\
&= (x_{i1}x_{j1} + x_{i2}x_{j2})^2 \\
&= (\mathbf{x_i}\mathbf{x'_j})^{\mathbf{2}} \\
&= \texttt{kernel}(\mathbf{x_i}, \mathbf{x_j})
\end{aligned}$$

i.e. even though the examples are in a higher-dimensional feature space, the values in the kernel matrix can be computed *only* using the examples with the original dimensionality! This is a *quadratic kernel*, with an explicit, finite-dimensional feature space. There are many different kernel functions with the same property, developed with many different applications in mind. I will describe just one more example, the Gaussian kernel (a.k.a. the Radial Basis Function, RBF, kernel). Informally, the result of using this kernel is similar to doing a locally weighted regression for a test example, where the prediction is a weighted combination of the training examples most similar to it.

The general definition for a RBF kernel is

$$\texttt{kernel}(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\frac{1}{2}(\mathbf{x_i} - \mathbf{x_j})\Sigma(\mathbf{x_i} - \mathbf{x_j})')$$

so a similarity function between examples that is akin to the Mahalanobis distance with a full covariance matrix. If the covariance matrix was restricted to being diagonal, with entries σ_j along the diagonal, this would become

$$\texttt{kernel}(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\frac{1}{2}\sum_{e=1}^{d}\frac{1}{\sigma_j^e}(x_{ie} - x_{je})^2)$$

so the similarity between examples becomes something like a squared euclidean distance where dimensions are weighted differently. And, finally, if the covariance matrix is restricted to being spherical, with a single σ in every element of the diagonal, we would have

$$\texttt{kernel}(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\frac{1}{2\sigma^2}\|\mathbf{x_i}, \mathbf{x_j}\|_2^2)$$

i.e. the similarity between examples is highest if their euclidean distance on the original space is small, and decays exponentially to 0 as they move apart, at a rate determined by σ. The RBF kernel is said to map to an infinite-dimensional space, since the implicit dimensionality scales with the number of examples (unlike the linear or quadratic kernels).

5 Bayesian linear regression

For this section I will assume a univariate target variable y, as it will simplify the presentation considerably; all the results generalize to multivariate targets. I also assume centered variables so that we can drop the bias term. is The notation is not always as rigorous as earlier, since main the goal is to provide the basic concepts prior to the reader undertaking any derivations. The model in this situation is

$$y_i = X_{i,:}\mathbf{w} + \epsilon_i$$

for each example $i = 1, \ldots, n$. Given that $\epsilon_i \sim N(0, \sigma^2)$, $y_i \sim N(X_{i,:}\mathbf{w}, \sigma^2)$. From this probabilistic perspective, the maximum likelihood estimate of the weights $\mathbf{w}$ is that which maximizes the likelihood

$$P(\mathbf{y}|X, \mathbf{w}, \sigma^2) = \Pi_{i=1}^{n} p(y_i|X_{i,:}, \mathbf{w}, \sigma^2)$$

which is the probability of all the training data, as a function of $\mathbf{w}$. This estimate is the same as what one would get from ordinary least squares. The estimation can be made analogous to ridge regression by specifying a particular prior distribution for the weights

$$\mathbf{w} \sim N(\mathbf{0}_d, \tau^2 I_d)$$

where each weight is independent and normally distributed around 0, and solving the resulting optimization problem (maximize likelihood above, but now multiplied by the prior). This is called a maximum *a posteriori* (MAP) estimate of $\mathbf{w}$, and the similarity to ridge regression comes from τ^2. If it is very small, the mass of the prior distribution of $\mathbf{w}$ is concentrated around 0; this would be akin to having a large regularization parameter λ in ridge regression, which would keep weights closer to 0.

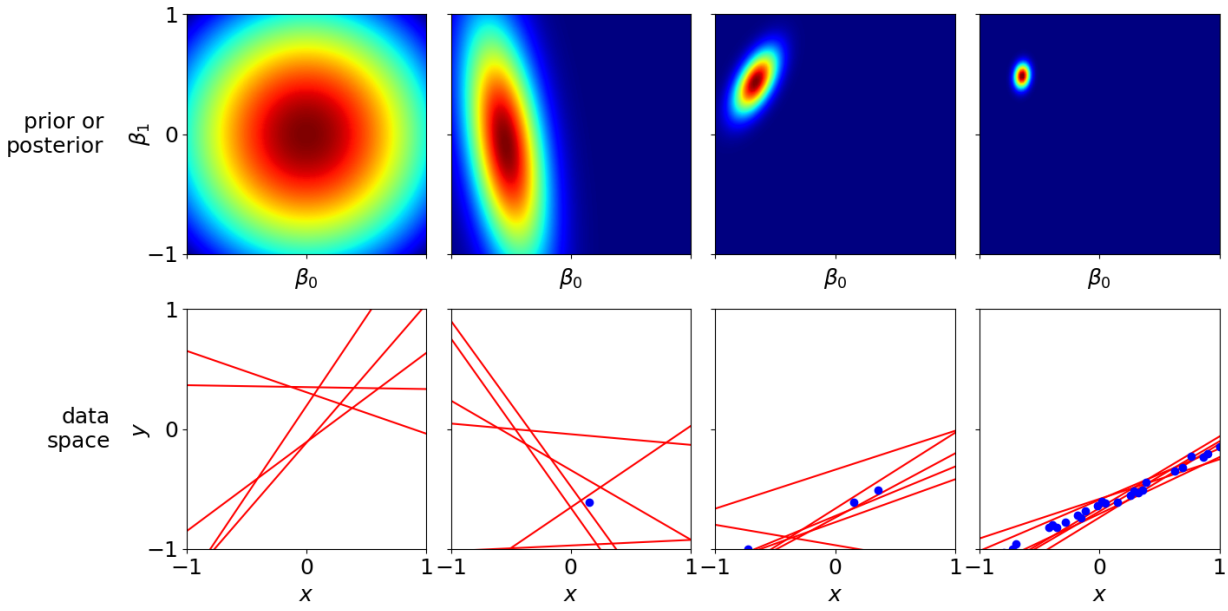

Figure 1: Rendering of Figure 3.7 from Bishop 2006, from `https://gregorygundersen.com/blog/2020/02/04/bayesian-linear-regression/`. The top row shows the probability density for the two weights of the model, β_0 and β_1, a priori (first plot), and a posteriori (other plots, as more data come in). The bottom row shows a few models – red lines – that one would get by sampling pairs of weights from the corresponding distribution in the first row. The blue dots are data points. As the number of data points increase, the posterior distribution gets much tighter, and the resulting model space is much more constrained.

This gives us more than a probabilistic analogue of ridge regression, however. If we expand the probability of the weights given data – the posterior distribution of w – we get

$$\begin{aligned} P(\mathbf{w}|\texttt{data}) &= P(\texttt{data}|\mathbf{w})P(\mathbf{w}) \\ &= N(\mathbf{y}|X, \mathbf{w}, \sigma^2)N(\mathbf{w}|\mathbf{0}_d, \tau^2 I_d) \\ &= N(\frac{1}{\sigma^2}A^{-1}X'\mathbf{y}, A^{-1}) \end{aligned}$$

where $A = \frac{1}{\sigma^2}X'X + \frac{1}{\tau^2}I_d$.

This is a probability distribution over all possible weight vectors, given the data we have seen and *a priori* expectations about how small they should be (determined by τ^2). Using a MAP estimate of the weights corresponds to using the most probable of those models, given the training data. Figure 1 shows what the posterior distribution of two weights on a regression model – β_0 and β_1 – look like as more data points are acquired, and also the resulting models from sampling a few weights.

Having this distribution makes it possible to generate a prediction for a test point using *all* possible models! This is akin to using an ensemble of models whose votes are weighted by their respective probability given the training data. This is called the *posterior predictive distribution* for a query point $\mathbf{x}_{test}$:

$$\begin{aligned} p(y_{test}|\mathbf{x}_{test}, D) &= \int_{\mathbf{w}} p(y_{test}|\mathbf{x}_{test}, \mathbf{w})p(\mathbf{w}|D)d\mathbf{w} \\ &= N\left(\frac{1}{\sigma^2}\mathbf{x}_{test}(\frac{1}{\sigma^2}X'X + \frac{1}{\tau^2}I)^{-1}X'\mathbf{y}, \mathbf{x}_{test}(\frac{1}{\sigma^2}X'X + \frac{1}{\tau^2})^{-1}\mathbf{x}'_{test} + \sigma^2\right) \end{aligned}$$

This approach has two potential advantages relative to using the MAP estimate. The first is that the mean of the distribution should be a more robust prediction. The second is that the standard deviation, being a function of the specific $\mathbf{x}_{test}$, acts as a measure of uncertainty about that prediction. In particular, it increases as $\mathbf{x}_{test}$ is further away from any of the training data points.

There are specific aspects that may vary across implementations. The first is the choice of prior, e.g. one could have used a normal–inverse–gamma prior as in Bishop 2006 instead of a normal. In both instances, the prior is a conjugate prior, i.e. the posterior distribution is in the same family as the prior, which makes it possible to solve the problem analytically. The second aspect is the way the hyperparameters – σ^2 and τ^2, in our example – are set. This can be done so that the prior is uninformative (we have no reason to prefer any value in the possible range), so that it includes known information, or simply to maximize some criterion on the training set $\mathbf{y}$ and X. The scikit-learn `BayesianRidge` package supports the latter.