# Note 6: Further Applications

Math 198: Math for Machine Learning

## Application: Ridge Regression

Our OLS proof requires the assumption that $\mathbf{X}$ is full rank. This is generally the case in practice; if drawing from a continuous data distribution, you will almost never observe colinear data. Nonetheless, if the data is nearly colinear, the input matrix will have singular values $\Sigma$ close to 0. This leads to numerical stability issues, as if $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ from SVD, then

$$(\mathbf{X}^\top\mathbf{X})^{-1} = (\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top)^{-1} = (\mathbf{V}\mathbf{\Sigma}\mathbf{I}\mathbf{\Sigma}\mathbf{V}^\top)^{-1}$$
$$= (\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top)^{-1} = (\mathbf{V}^\top)^{-1}\mathbf{\Sigma}^{-2}\mathbf{V}^{-1} = \mathbf{V}\mathbf{\Sigma}^{-2}\mathbf{V}^\top$$

As the smaller singular values in $\mathbf{\Sigma}$ decay to 0, the terms in $\mathbf{\Sigma}^{-2}$ grow very large, leading to numerical instability and large values in the output vector $\mathbf{w}$ which do not generalize well to unseen data. To prevent this from occurring, we can modify $\mathbf{X}^\top\mathbf{X}$ to ensure that it is full rank. Recall that $\mathbf{X}^\top\mathbf{X}$ is a positive semi-definite matrix. Therefore, for any scalar $\lambda$, $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}$ is positive definite and therefore full rank. So, we can substitute this quantity into the OLS solution to obtain a new solution to a new model:

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

This model is known as *ridge regression*. Note that the scalar $\lambda$ represents the magnitude of the perturbation to $\mathbf{X}^\top\mathbf{X}$, and is not a weight learned by the model. This *hyperparameter* is an input to the model, and must be set by the model's user. Typical values are between $10^{-4}$ and 1. We will soon show, using techniques of matrix calculus, that these weights solve the optimization problem

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

which is the OLS optimization problem, with an added penalty for the size of the weights determined by $\lambda$. Viewed in this sense, ridge regression prevents the weights from growing too large, avoiding generalization issues.

## Application: Total Least Squares

Recall that in our OLS framework, we assumed that there was noise in the observations, but not in the features; that is, the relationship between the features and the observations was modeled as

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \epsilon$$

where $\epsilon$ was a zero-mean normal random vector representing the noise in the observations. However, a more general approach would also assume noise in the features as well. We can find weights for this case; we model it as

$$(\mathbf{X} + \epsilon_{\mathbf{X}})\mathbf{w} = \mathbf{y} + \epsilon_{\mathbf{y}}$$

and now seek the weights which minimize the total error $||\epsilon_{\mathbf{X}}\ \epsilon_{\mathbf{y}}||^2$. This latter case is appropriately called *total least squares*. Note that geometrically and in two dimensions, OLS finds the line of best fit which minimizes *vertical* distance to the datapoints, whereas TLS finds the line of best fit which minimizes *Euclidean* distance to the datapoints. To do so, we rewrite the equation above as a matrix-vector product:

$$\begin{bmatrix}\mathbf{X} + \epsilon_{\mathbf{X}} & \mathbf{y} + \epsilon_{\mathbf{y}}\end{bmatrix}\begin{bmatrix}\mathbf{w} \\ -1\end{bmatrix} = 0$$

It is now much clearer that the vector $\begin{bmatrix}\mathbf{w} & -1\end{bmatrix}^\top$ is in the nullspace of $\begin{bmatrix}\mathbf{X}+\epsilon_\mathbf{X} & \mathbf{y}+\epsilon_\mathbf{y}\end{bmatrix}$. If this matrix is full-rank, then it has a trivial nullspace which cannot contain a non-zero vector; therefore, we must set the errors $\epsilon_\mathbf{x}, \epsilon_\mathbf{y}$ in such a way that the resulting matrix is not full-rank. To do so, we will make use of the low-rank approximation theorem from note 5. Let $d$ be the rank of the matrix $\begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix}$, which we assume is full-rank. Furthermore, let

$$\begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix} = \sum_{i=1}^{d} \mathbf{u}_i \sigma_I \mathbf{v}_i^\top$$

be the SVD of $\begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix}$. Then by the theorem from note 5, the best rank-$(d-1)$ approximation is given by

$$\begin{bmatrix}\mathbf{X}+\epsilon_\mathbf{X} & \mathbf{y}+\epsilon_\mathbf{y}\end{bmatrix} = \sum_{i=1}^{d-1} \mathbf{u}_i \sigma_I \mathbf{v}_i^\top$$

Furthermore, recall that the $\mathbf{V}^\top$ term is an orthogonal matrix; therefore, its columns are pairwise orthonormal, and so its $d$-th column $\mathbf{v}_d$ is in the kernel of $\begin{bmatrix}\mathbf{X}+\epsilon_\mathbf{X} & \mathbf{y}+\epsilon_\mathbf{y}\end{bmatrix}$. Since the rank of $\begin{bmatrix}\mathbf{X}+\epsilon_\mathbf{X} & \mathbf{y}+\epsilon_\mathbf{y}\end{bmatrix}$ is $d-1$ and its image is of dimension $d$, its kernel must be of dimension 1, and so $\mathbf{v}_d$ in fact spans the kernel. So we have

$$\begin{bmatrix}\mathbf{w} \\ -1\end{bmatrix} = \alpha \mathbf{v}_d$$

and so to find $\mathbf{w}$ we simply find $\alpha$ such that the last component of $\alpha \mathbf{v}_d$ is -1.

## Application: Feature Augmentation

So far, we have only modeled linear relationships, in which the output is a linear transformation of the input. This means that the only curve we can fit to two-dimensional data is a straight line. However, it may be the case that the relationship we are trying to learn is not linear – consider how poor a linear model would perform when trying to fit to sinusoidal data, for example. To use OLS to fit nonlinear data, we can add additional features which have nonlinear relationships with the inputs, and use OLS to determine their relative weight in the output. To do so, we can define a function $\phi$: $\mathbb{R}^l \to \mathbb{R}^d$ known as a *feature map*, which adds additional non-linear features of the raw inputs to the data matrix. This process is known as *feature augmentation*. As an example, suppose we are trying to fit a quartic polynomial in one variable. Therefore, the underlying function we are trying to approximate is of the form

$$f(x) = a + bx + cx^2 + dx^3 + ex^4$$

We are supplied with a one-dimensional feature vector $\mathbf{x}$ storing all the input values $x_i$, and a one-dimensional observation vector $\mathbf{y}$ storing all the output values $y_i$. We'd like to approximate the constant values $a, b, c, d, e$, so we can add features representing $x^0, x^1, x^2, x^3, x^4$ for each value of $x$ in $\mathbf{x}$. So our feature map $\phi$ in this case is $x \mapsto [1, x, x^2, x^3, x^4]$. We then use the augmented datamatrix $\mathbf{\Phi}$ as our input to OLS, where $\mathbf{\Phi}_i = [1, x_i, x_i^2, x_i^3, x_i^4]$. Our weight vector is therefore

$$\mathbf{w} = (\mathbf{\Phi}^\top \mathbf{\Phi} + \lambda \mathbf{I})^{-1} \mathbf{\Phi}^\top \mathbf{y}$$

using ridge regression. The weight vector contains our desired predictions $w_1 \approx a$, $w_2 \approx b$, etc.

Note that we will often augment our data by adding a column of all 1s, known as a *bias vector*. In two dimensions, this has the effect of the outputted prediction, a line, to have a y-intercept other than 0, i.e. not pass through the origin.

## Application: Kernel Trick

Our OLS solution contained an $\mathbf{X}^\top\mathbf{X}$ term, which has dimension $d \times d$, where $d$ is the number of features/-columns in the data matrix $\mathbf{X}$. However, if we are working with high-dimensional data, with significantly more features than datapoints, it may be desirable to compute with $\mathbf{X}\mathbf{X}^\top$, which has dimension $n \times n$, where $n$ is the number of datapoints/rows in the data matrix. Why is $\mathbf{X}\mathbf{X}^\top$ preferable for computation? The runtime to calculate $\mathbf{X}^\top\mathbf{X}$ for an $n \times d$ matrix $\mathbf{X}$ is $O(d^{2.3})$, whereas $\mathbf{X}\mathbf{X}^\top$ is $O(n^{2.3})$, using the fastest possible methods. The subsequent inversions have similar runtimes. Therefore, we can significantly speed up computation by working with $\mathbf{X}\mathbf{X}^\top$ when $d > n$. This situation can usually be avoided by collecting more data, but feature augmentation can often lead to situations with high numbers of features. Consider multivariable polynomials. A degree-5 polynomial in five variables has 252 terms, each of which has an associated weight. As the degree and number of variables grow, the number of terms grows quickly. To avoid our computation runtime from growing as well, let's attempt to find another equation from our OLS solution which expresses $\mathbf{w}$ in terms of $\mathbf{X}\mathbf{X}^\top$ instead of $\mathbf{X}^\top\mathbf{X}$.

The derivation follows from singular value decomposition, assuming $\mathbf{X}$ is full rank:

$$\begin{aligned}
\mathbf{w} &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} \\
&= (\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top)^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{y} \\
&= \mathbf{V}\mathbf{\Sigma}^{-2}\mathbf{V}^\top\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{y} \\
&= \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^\top\mathbf{y} \\
&= \mathbf{V}\mathbf{I}\mathbf{\Sigma}^{-1}\mathbf{U}^\top\mathbf{y} \\
&= \mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}^{-2}\mathbf{U}^\top\mathbf{y} \\
&= \mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top(\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top)^{-1}\mathbf{y} \\
&= \mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y}
\end{aligned}$$

The same numerical stability issues occur with this new solution, as $(\mathbf{X}\mathbf{X}^\top)^{-1} = \mathbf{U}\mathbf{\Sigma}^{-2}\mathbf{U}^\top$. Luckily, we have an alternate formation for ridge regression as well:

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

The proof of this one is more difficult, and is included for reference as an appendix.

So, we can compute with this version in the case of feature augmentation. Can we improve our runtime further? Yes, as it turns out – because of a special property of the $\mathbf{\Phi}\mathbf{\Phi}^\top$ term. Recall that the $i$-th row of $\mathbf{\Phi}$ is the feature map $\phi$ applied to the $i$-th row of $\mathbf{X}$, the raw data matrix. Then we can reformulate $\mathbf{\Phi}\mathbf{\Phi}^\top$ as

$$\mathbf{\Phi}\mathbf{\Phi}^\top = \begin{bmatrix} — & \phi(\mathbf{x_1})^\top & — \\ — & \phi(\mathbf{x_2})^\top & — \\ & \vdots & \\ — & \phi(\mathbf{x_n})^\top & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ \phi(\mathbf{x_1}) & \phi(\mathbf{x_2}) & \cdots & \phi(\mathbf{x_n}) \\ | & | & & | \end{bmatrix} = \begin{bmatrix} \phi(\mathbf{x_1})^\top\phi(\mathbf{x_1}) & \phi(\mathbf{x_1})^\top\phi(\mathbf{x_2}) & \cdots \\ \phi(\mathbf{x_2})^\top\phi(\mathbf{x_1}) & \ddots & \\ \vdots & & \phi(\mathbf{x_n})^\top\phi(\mathbf{x_n}) \end{bmatrix}$$

Let $k(\mathbf{x_i}, \mathbf{x_j}) = \langle\phi(\mathbf{x_i}), \phi(\mathbf{x_j})\rangle$. Then $(\mathbf{\Phi}\mathbf{\Phi}^\top)_{ij} = k(\mathbf{x_i}, \mathbf{x_j})$. The function $k$ thus takes raw-feature inputs and outputs their inner product in the feature space. Such a function is known as a *kernel function*. We refer to the associated matrix $\mathbf{\Phi}\mathbf{\Phi}^\top$ as the *Gram matrix*, and denote it as $\mathbf{K}(\{\mathbf{x_1} \ldots \mathbf{x_n}\})$. There are two equivalent definitions for a kernel function. A function $k(\mathbf{x_i}, \mathbf{x_j})$ is a kernel function if there exists a feature map $\phi$ such that $k(\mathbf{x_i}, \mathbf{x_j}) = \langle\phi(\mathbf{x_i}), \phi(\mathbf{x_j})\rangle$. Alternatively, if the Gram matrix $\mathbf{K}(\{\mathbf{x_1}, \ldots, \mathbf{x_n}\})$ is PSD for all $\{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$), then the associated function $k$ is a kernel function. Conveniently, the linear combination of any number of kernel functions is a valid kernel function. The $\mathbf{\Phi}\mathbf{\Phi}^\top$ solution is thus referred to as the *kernelized* formulation.

Using the kernelized formulation to speed up computation works for any feature map. We now encounter the *kernel trick*, which further speeds up computation of the $\mathbf{\Phi\Phi}^\top$ term when the feature map takes the raw input to a polynomial in the inputs. Consider the case of a degree-2 polynomial in two variables. The rows of our raw data matrix are therefore $\mathbf{x_i} = \begin{bmatrix} a_i & b_i \end{bmatrix}$, where $a_i = x_{i1}$ and $b_i = x_{i2}$. The feature map can be defined as

$$\phi(\mathbf{x_i}) = \begin{bmatrix} a_i^2 & b_i^2 & \sqrt{2}a_i b_i & \sqrt{2}a_i & \sqrt{2}b_i & 1 \end{bmatrix}^\top$$

and the kernel function becomes

$$\begin{aligned} k(\mathbf{x_i}, \mathbf{x_j}) &= \phi(\mathbf{x_i})^\top \phi(\mathbf{x_j}) \\ &= \begin{bmatrix} a_i^2 & b_i^2 & \sqrt{2}a_i b_i & \sqrt{2}a_i & \sqrt{2}b_i & 1 \end{bmatrix}^\top \begin{bmatrix} a_j^2 & b_j^2 & \sqrt{2}a_j b_j & \sqrt{2}a_j & \sqrt{2}b_j & 1 \end{bmatrix} \\ &= a_i^2 a_j^2 + b_i^2 b_j^2 + 2a_i b_i a_j b_j + 2a_i a_j + 2b_i b_j + 1 \\ &= (a_i a_j + b_i b_j)^2 + 2(a_i a_j + b_i b_j) + 1 \\ &= (\mathbf{x_i}^\top \mathbf{x_j})^2 + 2(\mathbf{x_i}^\top \mathbf{x_j}) + 1 \\ &= (\mathbf{x_i}^\top \mathbf{x_j} + 1)^2 \end{aligned}$$

In fact, for a $p$-dimensional polynomial with such a feature map, we have that $k_p(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i}^\top \mathbf{x_j} + 1)^p$. The kernel trick therefore reduces the complexity of computing the entries of $\mathbf{\Phi\Phi}^\top$ – no matter the degree $p$, the runtime is dependent only on the dimension of the raw feature space.

# Appendix: Kernelized Ridge Regression Proof

We start by manipulating the ridge regression solution to determine that $\mathbf{w} \in \text{range}(\mathbf{X}^\top)$:

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

$$(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = \mathbf{X}^\top\mathbf{y}$$

$$\mathbf{X}^\top\mathbf{X}\mathbf{w} + \lambda\mathbf{w} = \mathbf{X}^\top\mathbf{y}$$

$$\lambda\mathbf{w} = \mathbf{X}^\top\mathbf{y} - \mathbf{X}^\top\mathbf{X}\mathbf{w}$$

$$\mathbf{w} = \frac{\mathbf{X}^\top\mathbf{y} - \mathbf{X}^\top\mathbf{X}\mathbf{w}}{\lambda}$$

$$\mathbf{w} = \mathbf{X}^\top\frac{\mathbf{y} - \mathbf{X}\mathbf{w}}{\lambda}$$

Therefore, $\mathbf{w} = \mathbf{X}^\top\mathbf{v}$ for some vector $\mathbf{v} \in \mathbb{R}^n$. To find $\mathbf{w}$, we just need to find $\mathbf{v}$ such that $\mathbf{w} = \mathbf{X}^\top\mathbf{v}$. Suppose we have such $\mathbf{v}$. Then, proceeding from the third line of the above section,

$$\mathbf{X}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{v}) + \lambda(\mathbf{X}^\top\mathbf{v}) = \mathbf{X}^\top\mathbf{y}$$

$$\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top\mathbf{v} + \lambda\mathbf{v}) = \mathbf{X}^\top(\mathbf{y})$$

Note that if we had some $\mathbf{v}^*$ such that

$$\mathbf{X}\mathbf{X}^\top\mathbf{v}^* + \lambda\mathbf{v}^* = \mathbf{y}$$

then said $\mathbf{v}^*$ would also satisfy

$$\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top\mathbf{v} + \lambda\mathbf{v}) = \mathbf{X}^\top(\mathbf{y})$$

and would thus be a solution. Because $\mathbf{X}\mathbf{X}^\top = (\mathbf{X}^\top)^\top\mathbf{X}^\top$ and thus is PSD, $\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}$ is positive definite and thus invertible. So, we have $\mathbf{v}^* = (\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})^{-1}\mathbf{y}$, and thus

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