CS 189 Introduction to Machine Learning
Fall 2017 Note 19

1 Fundamental Theorem of Linear Algebra

Before we begin our discussion of kernels, let's first introduce the **Fundamental Theorem of Linear Algebra (FTLA)**. Suppose that there is a matrix (linear map) A that maps $\mathbb{R}^n$ to $\mathbb{R}^m$. Denote $\mathcal{N}(A)$ as the nullspace of A, and $\mathcal{R}(A)$ as the range of A. Then the following properties hold:

1. $\mathcal{N}(A) \overset{\perp}{\oplus} \mathcal{R}(A^T) = \mathbb{R}^n$

   The symbol $\oplus$ indicates that we taking a **direct sum** of $\mathcal{N}(A)$ and $\mathcal{R}(A^T)$, which means that $\forall u \in \mathbb{R}^n$ there exist unique elements $u_1 \in \mathcal{N}(A)$ and $u_2 \in \mathcal{R}(A^T)$ such that $u = u_1 + u_2$. Furthermore, the symbol $\perp$ indicates that $\mathcal{N}(A)$ and $\mathcal{R}(A^T)$ are orthogonal subspaces.

   Proof. We can equivalently prove that $\mathcal{N}(A) = \mathcal{R}(A^T)^\perp$.

   $$\begin{aligned} u \in \mathcal{N}(A) &\iff Au = 0 \\ &\iff \langle v, Au \rangle = 0 \quad \forall v \in \mathbb{R}^m \\ &\iff \langle A^T v, u \rangle = 0 \quad \forall v \in \mathbb{R}^m \\ &\iff u \in \mathcal{R}(A^T)^\perp \end{aligned}$$

   □

   $\mathcal{N}(A^T) \overset{\perp}{\oplus} \mathcal{R}(A) = \mathbb{R}^m$, by symmetry.

2. $\mathcal{N}(A^T A) = \mathcal{N}(A)$

   Proof. Let's prove $\mathcal{N}(A) \subseteq \mathcal{N}(A^T A)$ and $\mathcal{N}(A^T A) \subseteq \mathcal{N}(A)$:

   $$\begin{aligned} u \in \mathcal{N}(A) &\iff Au = 0 \\ &\implies A^T Au = 0 \\ &\iff u \in \mathcal{N}(A^T A) \end{aligned}$$

   $$\begin{aligned} u \in \mathcal{N}(A^T A) &\iff A^T Au = 0 \\ &\iff \langle u_1, A^T Au \rangle = 0 \quad \forall u_1 \in \mathbb{R}^n \\ &\iff \langle Au_1, Au \rangle = 0 \quad \forall u_1 \in \mathbb{R}^n \\ &\implies \langle Au, Au \rangle = 0 \\ &\iff Au = 0 \\ &\iff u \in \mathcal{N}(A) \end{aligned}$$

□

$\mathcal{N}(AA^T) = \mathcal{N}(A^T)$, by symmetry.

3. $\mathcal{R}(A^TA) = \mathcal{R}(A^T)$

Proof.

$$\begin{aligned} u \in \mathcal{R}(A^T) &\iff \exists v \in \mathbb{R}^m \quad A^T v = u \\ &\iff \exists v_1 \in \mathcal{R}(A), v_2 \in \mathcal{N}(A^T) \quad A^T(v_1 + v_2) = u \\ &\iff \exists v_1 \in \mathcal{R}(A) \quad A^T v_1 = u \\ &\iff \exists u_1 \in \mathbb{R}^n \quad A^T(Au_1) = u \\ &\iff u \in \mathcal{R}(A^TA) \end{aligned}$$

□

$\mathcal{R}(AA^T) = \mathcal{R}(A)$, by symmetry.

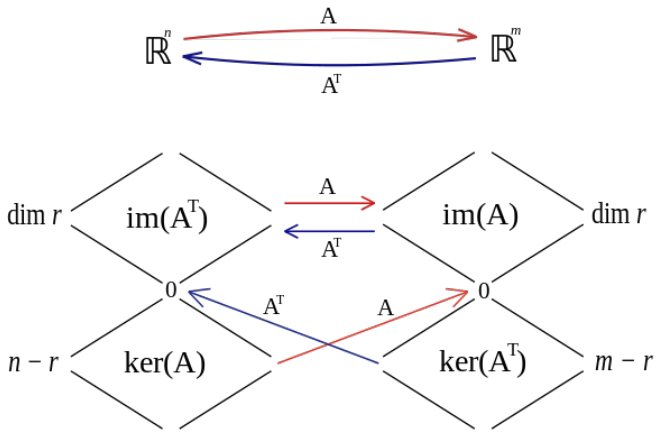

Figure 1: This diagram follows from the properties of FTLA

What exactly is the fundamental theorem of linear algebra useful for in the context of kernels? In most machine learning problems such as regression and SVM, we given a vector $y \in \mathbb{R}^n$ and a matrix $X \in \mathbb{R}^{n \times m}$, where n is the number of training points and m is the dimension of the raw data points. In most settings we don't want to work with just the raw feature space, so we augment features to the data points and end up with a matrix $A \in \mathbb{R}^{n \times d}$, where $a_i = \phi(x_i) \in \mathbb{R}^d$. Then we solve a well-defined optimization problem that involves A and y, over the weights $w \in \mathbb{R}^d$. Note the problem that arises here. If we have polynomial features of degree at most p in the raw m dimensional space, then there are $d = O(m^p)$ terms that we need to optimize, which can be very very very large, in fact much larger than the number of training points n. Wouldn't it be useful, if instead of solving a optimization problem over d variables, we could solve an equivalent problem

over (potentially much smaller) n variables? Here's where FTLA comes in. We know that we can express any $w \in \mathbb{R}^d$ as a unique combination $w = w_1 + w_2$, where $w_1 \in \mathscr{R}(A^T)$ and $w_2 \in \mathscr{N}(A)$. Equivalently we can express this as $w = A^T v + r$, where $v \in \mathbb{R}^n$ and $r \in \mathscr{N}(A)$. Now, instead of optimizing over $w \in \mathbb{R}^d$, we can optimize over $v \in \mathbb{R}^n$ and $r \in \mathbb{R}^d$, which equates to optimizing over $n+d$ variables. However, as we shall see, the optimization over r will be trivial so we just have to optimize an n dimensional problem. Let's first see how this plays our in ridge regression.

2 Ridge Regression

We know that $w = A^T v + r$, where $v \in \mathbb{R}^n$ and $r \in \mathscr{N}(A)$. Let's now solve ridge regression by optimizing over the variables v and r instead of w:

$$\begin{aligned}
v^*, r^* &= \underset{v,r \in \mathscr{N}(A)}{\arg\min} \; ||Aw - y||_2^2 + \lambda ||w||_2^2 \\
&= \underset{v,r \in \mathscr{N}(A)}{\arg\min} \; ||A(A^T v + r) - y||_2^2 + \lambda ||A^T v + r||_2^2 \\
&= \underset{v,r \in \mathscr{N}(A)}{\arg\min} \; ||AA^T v + \cancel{Ar} - y||_2^2 + \lambda ||A^T v + r||_2^2 \\
&= \underset{v,r \in \mathscr{N}(A)}{\arg\min} \; \left(v^T AA^T AA^T v - 2v^T AA^T y + y^T y\right) + \lambda \left(v^T AA^T v + \cancel{2v^T Ar} + r^T r\right) \\
&= \underset{v,r \in \mathscr{N}(A)}{\arg\min} \; \left(v^T AA^T AA^T v - 2v^T AA^T y\right) + \lambda \left(v^T AA^T v + r^T r\right)
\end{aligned}$$

We crossed out Ar and $2v^T Ar$ because $r \in \mathscr{N}(A)$ and therefore $Ar = 0$. Now we are optimizing over $L(v,r)$, which is **jointly convex** in v and r, because its hessian is positive semi-definite. Let's show that this is indeed the case:

$$\begin{aligned}
\nabla_r^2 L(v,r) &= 2I \succeq 0 \\
\nabla_r \nabla_v L(v,r) &= \nabla_v \nabla_r L(v,r) = 0 \\
\nabla_v^2 L(v,r) &= 2AA^T AA^T + 2\lambda AA^T \succeq 0
\end{aligned}$$

Since the cross terms of the hessian are 0, it suffices that $\nabla_r^2 L(v,r)$ and $\nabla_v^2 L(v,r)$ are psd to establish joint convexity. With joint convexity established, we can set the gradient to 0 wrt r and v and obtain the global minimum:

$$\nabla_r L(v,r) = 2r = 0 \implies r^* = 0$$

Note that $r^* = 0$ just so happens in to be in $\mathscr{N}(A)$, so it is a feasible point.

$$\begin{aligned}
\nabla_v L(v,r) = 2AA^T AA^T v - 2AA^T y + 2\lambda AA^T v &= 0 \\
\implies AA^T (AA^T + \lambda I) v &= AA^T (y) \\
\implies v^* &= (AA^T + \lambda I)^{-1} y
\end{aligned}$$

Note that $AA^T + \lambda I$ is positive definite and therefore invertible, so we can compute $(AA^T + \lambda I)^{-1} y$. Even though $(AA^T + \lambda I)^{-1} y$ is a critical point for which the gradient is 0, it must achieve the global minimum because the objective is jointly convex. We conclude that

$$w^* = A^T (AA^T + \lambda I)^{-1} y$$

Recall that previously, we derived ridge regression and ended up with

$$w^* = (A^T A + \lambda I)^{-1} A y$$

In fact, these two are equivalent expressions! The question that now arises is which expression should you pick? Which is *more efficient* to calculate? We will answer this question after we introduce kernels.

2.1 Alternative Derivation

We can arrive at the same expression for w^* with some clever algebraic manipulations. Our previous derivation of ridge regression was $w^* = (A^T A + \lambda I)^{-1} A^T Y$. Rearranging the terms of this equation, we have

$$\begin{aligned}
(A^T A + \lambda I) w &= A^T y \\
A^T A w + \lambda w &= A^T y \\
\lambda w &= A^T y - A^T A w \\
w &= \frac{1}{\lambda}(A^T y - A^T A w) \\
w &= \frac{A^T y - A^T A w}{\lambda} \\
w &= A^T \frac{y - A w}{\lambda}
\end{aligned}$$

which says that *whatever w is*, it is some linear combination of the training points a_i (because anything of the form $A^T v$ is a linear combination of the columns of A^T, which are the training points). To find w it suffices to find v, where $w = A^T v$.

Recall that the relationship we have to satisfy is $A^T A w - \lambda w = A^T y$. Let's assume that we had v, and just substitute $A^T v$ in for all the w's.

$$\begin{aligned}
A^T A (A^T v) + \lambda (A^T v) &= A^T y \\
A^T A A^T v + A^T (\lambda v) &= A^T y \\
A^T (A A^T v + \lambda v) &= A^T (y)
\end{aligned}$$

We can't yet isolate v and have a closed-form solution for it, but we *can* make the observation that if we found an v such that we had

$$A A^T v + \lambda v = y$$

that would *imply* that this v also satisfies the above equation. Note that we did not "cancel the A^T's on both sides of the equation." We saw that having v satisfy one equation implied that it satisfied the other as well. So, indeed, we can isolate v in this new equation:

$$(A A^T + \lambda I) v = y \implies v^* = (A A^T + \lambda I)^{-1} y$$

and have that the v which satisfies this equation will be such that $A^T v$ equals w. We conclude that the optimal w is

$$w^* = A^T v^* = A^T (A A^T + \lambda I)^{-1} y$$

3 Kernels

Recall that in the solution for ridge regression we have an AA^T term. If we expand this term we have that

$$AA^T = \begin{pmatrix} — & a_1^T & — \\ — & a_2^T & — \\ & \vdots & \\ — & a_n^T & — \end{pmatrix} \begin{pmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_n \\ | & | & & | \end{pmatrix} = \begin{pmatrix} a_1^T a_1 & a_1^T a_2 & \dots \\ a_2^T a_1 & \ddots & \\ \vdots & & a_n^T a_n \end{pmatrix}$$

Each entry AA^T_{ij} is a dot product between a_i and a_j and can be interpreted as a similarity measure. In fact, we can express

$$AA^T_{ij} = \langle a_i, a_j \rangle = \langle \phi(x_i), \phi(x_j) \rangle = k(x_i, x_j)$$

where $k(.,.)$ is the **kernel** function. The kernel function takes raw-feature inputs and outputs their inner product in the augmented feature space. We denote the matrix of $k(x_i, x_j)$ terms as the **Gram matrix** and denote it as K:

$$K = AA^T = \begin{pmatrix} k(x_1,x_1) & k(x_1,x_2) & \dots \\ k(x_2,x_1 & \ddots & \\ \vdots & & k(x_n,x_n) \end{pmatrix}$$

Formally, $k(x_1, x_2)$ is defined to be a valid kernel function if there exists a feature map $\phi(.)$ such that $\forall x_1, x_2$,

$$k(x,y) = \langle \phi(x_1), \phi(x_2) \rangle$$

Equivalently, we can also say that for all sets $\mathscr{D} = \{a_1, a_2, \dots, a_n\}$, the Gram matrix $K(\mathscr{D})$ is positive semi-definite.

Computing the each Gram matrix entry $k(x_i, x_j)$ can be done in a straightforward fashion if we apply the feature map to x_i and x_j and then take their dot product in the augmented feature space — this takes $O(d)$ time, where d is the dimensionality of the problem in the augmented feature space. However, if we use the **kernel trick**, we can perform this operation in $O(m + \log p)$ time, where m is the dimensionality of the problem in the raw feature space and k is the degree of the polynomials in the augmented feature space.

4 The Kernel Trick

Suppose that you are computing $k(x,z)$, using a p-degree polynomial feature map that maps m dimensional inputs to $d = O(m^p)$ dimensional outputs. Let's take $p = 2$ and $m = 2$ as an example.

We have that

$$\begin{aligned}
k(x,z) &= \langle \phi(x), \phi(z) \rangle \\
&= \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}x_1x_2 & \sqrt{2}x_1 & \sqrt{2}x_2 & 1 \end{bmatrix}^T \begin{bmatrix} z_1^2 & z_2^2 & \sqrt{2}z_1z_2 & \sqrt{2}z_1 & \sqrt{2}z_2 & 1 \end{bmatrix} \\
&= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1z_1x_2z_2 + 2x_1z_1 + 2x_2z_2 + 1 \\
&= (x_1^2z_1^2 + 2x_1z_1x_2z_2 + x_2^2z_2^2) + 2x_1z_1 + 2x_2z_2 + 1 \\
&= (x_1z_1 + x_2z_2)^2 + 2(x_1z_1 + x_2z_2) + 1 \\
&= (x^Tz)^2 + 2x^Tz + 1 \\
&= (x^Tz + 1)^2
\end{aligned}$$

We can compute $k(x,z)$ either by

1. Raising the inputs to the augmented feature space and take their inner product
2. Computing $(x^Tz+1)^2$, which involves an inner product of the raw-feature inputs

Clearly, the latter option is much cheaper to calculate, taking $O(m+\log p)$ time, instead of $O(m^p)$ time. In fact, this concept generalizes for any arbitrary m and p, and for p-degree polynomial features, we have that

$$k(x,z) = (x^Tz+1)^p$$

The kernel trick makes computations significantly cheaper to perform, making kernelization much more appealing! The takeaway here is that no matter what the degree p is, the computational complexity is the same — it is only dependent on the dimensionality of the raw feature space!

5 Computational Analysis

Back to the original question – in ridge regression, should we compute

$$w^* = A^T(AA^T + \lambda I)^{-1}y$$

or

$$w^* = (A^TA + \lambda I)^{-1}Ay$$

Let's compare their computational complexities. Suppose you are given an arbitrary test point $z \in \mathbb{R}^m$, and you would like to compute its predicted value $\hat{y}$. Let's see how these values are calculated in each case:

1. Kernelized

   $$\hat{y} = \langle \phi(z), w^* \rangle = \phi(z)^TA^T(AA^T + \lambda I)^{-1}y = \begin{bmatrix} k(x_1,z) & \dots & k(x_n,z) \end{bmatrix}^T (K + \lambda I)^{-1}y$$

   Computing the K term takes $O(n^2(m+\log p))$, and inverting the matrix takes $O(n^3)$. These two computations dominate, for a total computation time of $O(n^3 + n^2(m+\log p))$.

2. Non-kernelized

$$\hat{y} = \langle \phi(z), w^* \rangle = \phi(z)^T (A^T A + \lambda I)^{-1} A^T y$$

Computing the $A^T A$ term takes $O(d^2 n)$, and inverting the matrix takes $O(d^3)$. These two computations dominate, for a total computation time of $O(d^3 + d^2 n) = O(m^{3p} + m^{2p} n)$.

Here is the takeaway: if $d << n$, the non-kernelized method is preferable. Otherwise if $n << d$, the kernelized method is preferable.

This discussion now motivates a major reason for dual SVMs. Recall that the dual SVM formulation is an optimization problem over n variables instead of d variables. It incorporates a $Q = (\text{diag } y) AA^T (\text{diag } y)$ term, and using the kernel trick, the AA^T term takes $O(n^2(m + \log p))$ instead of $O(n^2 m^p)$ to calculate, making the dual SVM formulation a very attractive choice when $n << d$.

Kernelization can be applied to pretty much any machine learning problem, such as logistic regression, k nearest neighbors, and even PCA.