CS 189 Introduction to Machine Learning
Fall 2019 Jennifer Listgarten & Stella Yu DIS5

1 Fun with least squares

Ordinary least squares for a scalar helps us learn linear predictors for a scalar b given observations $\mathbf{a}$. This is done by taking our training data points and constructing a tall vector $\mathbf{b}$ by stacking up our training data's b_i and a corresponding tall matrix A by stacking up our training data's $\mathbf{a}_i^T$, and then finding the weights $\mathbf{x}$ that minimize $\|A\mathbf{x} - \mathbf{b}\|_2^2$. The resulting weights $\mathbf{x}$ can be used to predict b's by $\mathbf{x}^T\mathbf{a}$.

Let us think about the components of $\mathbf{a}$ as being a sequence of measurements in time. The first measurement, the second measurement, the third measurement, and so on.

1. Suppose that you wanted to construct not one linear predictor for b but a sequence of them. One that just used the first measurement. One that used the first two measurements. All the way up to one that uses all the measurements. How would you do this in the most straightforward manner?

2. Someone suggests that maybe the measurements themselves are partially predictable from the previous measurements. They suggest a two part strategy. First we predict the next measurement based on the measurements so far. And then we look at the difference (sometimes deemed the "innovation") between the actual measurement we made and our prediction for it, and just use that to update our prediction for b.

   Give a way to learn (from the training data) these best predictors of the next measurement from the previous measurements as well as learn (from the training data) the weights used to update the prediction for b based on the innovation.

3. Is this two-part prediction strategy equivalent to the straightforward approach? Why or why not?

   HINT: Think about what it might mean to orthonormalize the columns of the training matrix A above.

4. **(BONUS — but worth doing)** A student complains to you that the case of learning a linear predictor for $\mathbf{b}$ given $\mathbf{a}$ seems too complicated. She suggests just setting up the problem directly as learning a matrix X so that $\mathbf{b} \approx X\mathbf{a}$. She takes the training data $(\mathbf{a}_i, \mathbf{b}_i)$ and stacks them **horizontally** into fat matrices A and B and computes the hypothetical difference matrix $B - XA$.

   She would like to minimize an appropriate norm for this and chooses the simple entry-wise 2-norm (also known as the Frobenius norm) squared. So $\min_X \|B - XA\|_F^2$. Recall that the Frobenius norm of a real matrix (need not be square) L is $\|L\|_F^2 = \mathrm{trace}(L^T L)$. Use this and vector calculus to directly solve for the minimizing X.

5. Comment on why it makes sense that the computation of the best linear prediction of a vector just turns into a set of best linear predictions of each scalar component of the vector being predicted.

2 Polynomial Kernels

Let ϕ be the quadratic feature map. For simplicity, let the original number of features be 2 (i.e., $x_i \in \mathbb{R}^2$).

1. Show that $(x^\top z + 1)^2 = \phi(x)^\top \phi(z)$.
2. What is the run-time of computing $(x^\top z + 1)^2$? How about the run-time of computing $\phi(x)^\top \phi(z)$ directly?