Math Companion to *The Art of Machine Learning*

Norm Matloff

University of California, Davis

This document is a supplement to *The Art of Machine Learning*, for readers who wish to know some of the mathematical underpinnings.

Contents

Chapter 1

Review of the Basics of Mathematical Probability

Listed below are the highlights of a standard calculus-based course in probability. We will mainly use only a few of them here — expected value, variance, covariance and the normal distribution family — but any reader interested in pursuing the mathematical side of ML should have all of these in their math toolkit.

- **expected value:**

  Consider random variables X and Y (not assumed independent), and constants c_1 and c_2. We have:

$$E(X+Y) = EX + EY \tag{1.1}$$

$$E(c_1 X) = c_1 EX \tag{1.2}$$

$$E(c_1 X + c_2 Y) = c_1 EX + c_2 EY \tag{1.3}$$

  By induction,

$$E(a_1 U_1 + ... + a_k U_k) = a_1 EX_1 + ... + a_k EX_k \tag{1.4}$$

  for random variables U_i and constants a_i.

  If X and Y *are* independent, then

$$E(XY) = EX \cdot EY \tag{1.5}$$

- **variance:**

  For any random variable W,

$$Var(W) = E[(W - EW)^2] = E(W^2) - (EW)^2 \tag{1.6}$$

  Consider random variables X and Y (now assumed independent), and constants c_1 and c_2. We have:

$$Var(X + Y) = Var(X) + Var(Y) \tag{1.7}$$

$$Var(c_1 X) = c_1^2 Var(X) \tag{1.8}$$

  By induction,

$$Var(a_1 U_1 + ... + a_k U_k) = a_1^2 Var(U_1) + ... + a_k^2 Var(U_k) \tag{1.9}$$

  for independent random variables U_i and constants a_i.

- **covariance:**

$$Cov(X, Y) = E[(X - EX)(Y - EY)] \tag{1.10}$$

  Roughly, how much do X and Y vary together? The scaled version,

$$\rho(X, Y) = \frac{Cov(X, Y)}{\sqrt{Var(X)} \cdot \sqrt{Var(Y)}} \tag{1.11}$$

  is the *correlation* betweeen X and Y. It takes on values in [-1,1]. When $X = Y$, then $Cov(X, Y) = Var(X)$ and the correlation is 1.0. But note that that is also true if $Y = cX + d$ for constants c and d with $c > 0$. (If $c < 0$, the correlation is -1.)

  For any constants c_1, c_2, c_3, c_4, we have

$$Cov(c_1 X + c_2 Y, c_3 X + c_4 Y) = c_1 c_2 Var(X) + c_3 c_4 Var(Y) + c_1 c_2 Cov(X, Y) \tag{1.12}$$

- **indicator random variables:**

  Equal 1 or 0, depending on whether a specified event A occurs.

  If T is an indicator random variable for the event A, then

$$ET = P(A), \;\; Var(T) = P(A)[1 - P(A)] \tag{1.13}$$

- **distributions:**

  - **cumulative distribution functions (cdfs):**
    For any random variable X,

$$F_X(t) = P(X \leq t), \quad -\infty < t < \infty \tag{1.14}$$

  - **probability mass functions (pmfs):**
    For a discrete random variable X (i.e. takes on finitely many or countably infinitely many values),

$$p_X(k) = P(X = k) \tag{1.15}$$

  - **density functions:**
    For a continuous random variable X (i.e. takes on a continuum of values), its probability density function is

$$f_X(t) = \frac{d}{dt} F_X(t), \quad -\infty < t < \infty \tag{1.16}$$

    and

$$P(X \text{ in } A) = \int_A f_X(s)\, ds \tag{1.17}$$

- **famous parametric families of distributions:**

  Just like one can have a family of curves, say $sin(2\pi n\theta(t))$ (different curve for each n and θ), certain families of distributions have been found useful. They're called *parametric families*, because they are indexed by one or more parameters, analogously to n and θ above.

  discrete:

  - **geometric:**
    Number of i.i.d. trials until first success. For success probability p:

$$p_N(k) = (1-p)^k p \tag{1.18}$$

$$EN = 1/p, \;\; Var(N) = \frac{1-p}{p^2} \tag{1.19}$$

– **binomial**

Number of successes in n i.i.d. trials, probability p of success per trial:

$$p_N(k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{1.20}$$

$$EN = np, \;\; Var(N) = np(1-p) \tag{1.21}$$

– **Poisson:**

Has often been found to be a good model for counts over time periods. Also used to model spatial counts.

One parameter, often called λ. Then

$$p_N(k) = \frac{e^{-\lambda}\lambda^k}{k!}, k = 0, 1, 2, ... \tag{1.22}$$

$$EN = Var(N) = \lambda \tag{1.23}$$

– **negative binomial:**

Number of i.i.d. trials until rth success. For success probability p:

$$p_N(k) = \binom{k-1}{r-1} (1-p)^{k-r} p^r, k = r, r+1, ... \tag{1.24}$$

$$E(N) = r \cdot \frac{1}{p}, \;\; Var(N) = r \cdot \frac{1-p}{p^2} \tag{1.25}$$

continuous:

– **uniform:**

All points "equally likely." If the interval is (q, r),

$$f_X(t) = \frac{1}{r-q}, \quad q < t < r \tag{1.26}$$

$$EX == \frac{q+r}{2}, \;\; Var(D) = \frac{1}{12}(r-q)^2 \tag{1.27}$$

– **normal (Gaussian):**

"Bell-shaped curves." Central Limit Theorem says, roughly, that the density of a sum of random variables is approximately normal.

Closed under affine transformations, i.e. if X is normally distributed, then so is $cX + d$ for any constants c and d.

Parameterized by mean and variance, μ and σ^2:

$$f_X(t) = \frac{1}{\sqrt{2\pi}\sigma}\, e^{-0.5\left(\frac{t-\mu}{\sigma}\right)^2}, -\infty < t < \infty \tag{1.28}$$

exponential:

- Memoryless! One parameter, usually called λ. Connectedt to Poisson family.

$$f_X(t) = \lambda e^{-\lambda t}, 0 < t < \infty \tag{1.29}$$

$$EX = 1/\lambda,\ \ Var(X) = 1/\lambda^2 \tag{1.30}$$

- **gamma:**
  Special case, Erlang family, arises as the distribution of the sum of i.i.d. exponential random variables.

$$f_X(t) = \frac{1}{\Gamma(r)}\lambda^r t^{r-1} e^{-\lambda t},\ t > 0 \tag{1.31}$$

- **iterated expected values:**
  - For discrete U ,

$$E(V) = \sum_c P(U = c)\ E(V \mid U = c) \tag{1.32}$$

  - For continuous V,

$$E(W) = \int_{-\infty}^{\infty} f_V(t)\ E(W \mid V = t)\ dt \tag{1.33}$$

Chapter 2

Basics of Linear Algebra

Machine learning (ML) techniques often make use of linear algebra, well beyond mere matrix multiplication. There are some issues that will come up frequently. We'll cover some of them briefly here, more later as the need arises. This chapter will review linear algebra, or serve as a quick treatment for those lacking this background. There will also be some topics that are new to most who have a good background.

2.1 Terminology and Notation

A *matrix* is a rectangular array of numbers. A *vector* is a matrix with only one row (a *row vector*) or only one column (a *column vector*). The default will be column vectors.

The expression, "the (i, j) element of a matrix," will mean its element in row i, column j.

If A is a *square* matrix, i.e., one with equal numbers n of rows and columns, then its *diagonal* elements are a_{ii}, $i = 1, ..., n$.

2.1.1 Matrix Addition and Multiplication

- For two matrices that have the same numbers of rows and same numbers of columns, addition is defined elementwise, e.g.

$$\begin{pmatrix} 1 & 5 \\ 0 & 3 \\ 4 & 8 \end{pmatrix} + \begin{pmatrix} 6 & 2 \\ 0 & 1 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} 7 & 7 \\ 0 & 4 \\ 8 & 8 \end{pmatrix} \tag{2.1}$$

- Multiplication of a matrix by a *scalar*, i.e., a number, is also defined elementwise, e.g.

$$0.4 \begin{pmatrix} 7 & 7 \\ 0 & 4 \\ 8 & 8 \end{pmatrix} = \begin{pmatrix} 2.8 & 2.8 \\ 0 & 1.6 \\ 3.2 & 3.2 \end{pmatrix} \tag{2.2}$$

- The *inner product* or *dot product* of equal-length vectors X and Y is defined by

$$\sum_{k=1}^{n} x_k y_k \tag{2.3}$$

- The product of matrices A and B is defined if the number of rows of B equals the number of columns of A. (A and B are then said to be *conformable*.) In that case, the (i, j) element of the product C is defined to be

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \tag{2.4}$$

  For instance,

$$\begin{pmatrix} 7 & 6 \\ 0 & 4 \\ 8 & 8 \end{pmatrix} \begin{pmatrix} 1 & 6 \\ 2 & 4 \end{pmatrix} = \begin{pmatrix} 19 & 66 \\ 8 & 16 \\ 24 & 80 \end{pmatrix} \tag{2.5}$$

  It is helpful to visualize c_{ij} as the inner product of row i of A and column j of B, e.g. as shown in bold face here:

$$\begin{pmatrix} \mathbf{7} & \mathbf{6} \\ 0 & 4 \\ 8 & 8 \end{pmatrix} \begin{pmatrix} \mathbf{1} & 6 \\ \mathbf{2} & 4 \end{pmatrix} = \begin{pmatrix} \mathbf{19} & 66 \\ 8 & 16 \\ 24 & 80 \end{pmatrix} \tag{2.6}$$

- Matrix multiplication is associative and distributive, but in general not commutative:

$$A(BC) = (AB)C \tag{2.7}$$

$$A(B + C) = AB + AC \tag{2.8}$$

$$AB \neq BA \tag{2.9}$$

2.2 Matrix Transpose

- The transpose of a matrix A, denoted A' or A^T, is obtained by exchanging the rows and columns of A, e.g.

$$\begin{pmatrix} 7 & 70 \\ 8 & 16 \\ 8 & 80 \end{pmatrix}' = \begin{pmatrix} 7 & 8 & 8 \\ 70 & 16 & 80 \end{pmatrix} \tag{2.10}$$

- If $A + B$ is defined, then

$$(A + B)' = A' + B' \tag{2.11}$$

- If A and B are conformable, then

$$(AB)' = B'A' \tag{2.12}$$

To save space on the page, we will usually write a column vector as the transpose of a row vector. For example, we will write

$$v = (3, 8)' \tag{2.13}$$

rather than

$$\begin{pmatrix} 3 \\ 8 \end{pmatrix} \tag{2.14}$$

2.3 Matrix Inverse

- The *identity* matrix I of size n has 1s in all of its diagonal elements but 0s in all off-diagonal elements. It has the property that $AI = A$ and $IA = A$ whenever those products are defined.
- If A is a square matrix and $AB = I$, then B is said to be the *inverse* of A, denoted A^{-1}. Then $BA = I$ will hold as well.
- If A and B are square, conformable and invertible, then AB is also invertible, and

$$(AB)^{-1} = B^{-1}A^{-1} \tag{2.15}$$

2.4 Eigenvalues and Eigenvectors

Let A be a square matrix.[1]

- A scalar λ and a nonzero vector X that satisfy

$$AX = \lambda X \tag{2.16}$$

  are called an *eigenvalue* and *eigenvector* of A, respectively. Note that for a given eigenvalue λ and eigenvector X, the vector cX is also an eigenvector for that eigenvalue, for any $c \neq 0$.

- If A is symmetric and real, then it is *diagonalizable*, i.e., there exists a matrix U such that

$$U'AU = D \tag{2.17}$$

  for a diagonal matrix D. The elements of D are the eigenvalues of A, and the columns of U are the eigenvectors of A (scaled to have length 1). Also, the eigenvectors will be *orthogonal*, meaning the inner product of any pair of them will be 0.

2.5 Matrix Rank and Vector Linear Independence

2.5.1 Linear Independence

Consider the matrix

$$M = \begin{pmatrix} 1 & 5 & 1 & -2 \\ 8 & 3 & 2 & 8 \\ 9 & 8 & 3 & 6 \end{pmatrix} \tag{2.18}$$

Note that the third row is the sum of the first two. In many contexts, this would imply that there are really only two "independent" rows in M in some sense related to the application.

Denote the rows of M by $r_i,\ i = 1, 2, 3$. We say they are *linearly independent* if it is not possible to find scalars a_i, at least one of them nonzero, such that the *linear combination* $a_1 r_1 + a_2 r_2 + a_3 r_3$ is equal to 0. In this case $a_1 = a_2 = 1$ and $a_3 = -1$ gives us 0, so the rows of M are linearly dependent.

The *rank* of a matrix is its maximal number of linearly independent rows or columns. The rank of M above is 2.

[1] For nonsquare matrices, the discussion here would generalize to the topic of *singular value decomposition*.

The reason we say "rows or columns" above is that it can be shown that the row rank and column rank are the same. Note that the implies that the rank of an $r \times s$ matrix is thus at most $\min(r, s)$. In the case of equality, we say the matrix has *full rank*.

2.5.2 Vector Spaces

A set of vectors $\mathcal{V}$ is called a *vector space* if:

- If X and Y are in $\mathcal{V}$, then so is $X + Y$.
- If X is in $\mathcal{V}$, then so is cX for any scalar c.

An important concept is that the *basis* of a vector space $\mathcal{V}$. It is a linearly independent set of vectors whose linear combinations collectively form all of $\mathcal{V}$. Here r_1 and r_2 form a basis for the *row space* of M. Alternatively, r_1 and r_3 also form a basis, as do r_2 and r_3.

2.6 Partitioned Matrices

It is often helpful to partition a matrix into *blocks* (often called *tiles* in the parallel computation community).

2.6.1 How It Works

Consider matrices A, B and C,

$$A = \begin{pmatrix} 1 & 5 & 12 \\ 0 & 3 & 6 \\ 4 & 8 & 2 \end{pmatrix} \tag{2.19}$$

and

$$B = \begin{pmatrix} 0 & 2 & 5 \\ 0 & 9 & 10 \\ 1 & 1 & 2 \end{pmatrix}, \tag{2.20}$$

so that

$$C = AB = \begin{pmatrix} 12 & 59 & 79 \\ 6 & 33 & 42 \\ 2 & 82 & 104 \end{pmatrix}. \tag{2.21}$$

We could partition A as, say,

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \tag{2.22}$$

where

$$A_{11} = \begin{pmatrix} 1 & 5 \\ 0 & 3 \end{pmatrix}, \tag{2.23}$$

$$A_{12} = \begin{pmatrix} 12 \\ 6 \end{pmatrix}, \tag{2.24}$$

$$A_{21} = \begin{pmatrix} 4 & 8 \end{pmatrix} \tag{2.25}$$

and

$$A_{22} = \begin{pmatrix} 2 \end{pmatrix}. \tag{2.26}$$

Similarly we would partition B and C into blocks of a compatible size to A,

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \tag{2.27}$$

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}, \tag{2.28}$$

so that for example

$$B_{21} = \begin{pmatrix} 1 & 1 \end{pmatrix}. \tag{2.29}$$

The key point is that multiplication still works if we pretend that those submatrices are numbers! For example, pretending like that would give the relation

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}, \tag{2.30}$$

which the reader should verify really is correct as matrices, i.e. the computation on the right side really does yield a matrix equal to C_{11}.

2.6.2 Important Special Case: Matrix Times Vector

Consider the product of a matrix and a vector, Ax. This product is a linear combination of the columns of A. To see this, write

$$A = (A_1, A_2, A_3) \tag{2.31}$$

with A_i being the i^{th} column, and

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{2.32}$$

The point is that (2.31) looks like a row vector — it isn't, but we pretend it is — so that Ax looks like the "dot product of one vector with another. That would give us

$$Ax = x_1A_1 + x_2A_2 + x_3A_3 \tag{2.33}$$

As noted, we then "unpretend," and find that (2.33) says that

> Ax is a linear combination of the columns of A. The coefficients in that linear combination are the elements of x.

2.6.3 Approximate Matrix Factorization

Given a matrix A, some ML algorithms attempt to find an "approximate factorization,"

$$A \approx WH \tag{2.34}$$

where A is $r \times s$, W is $r \times m$ and H is $m \times s$. Partition the first and third matrices into rows, i.e. write

$$A = \begin{pmatrix} a_1 \\ ... \\ a_r \end{pmatrix}, \tag{2.35}$$

and

$$H = \begin{pmatrix} h_1 \\ ... \\ h_m \end{pmatrix}, \tag{2.36}$$

so that for instance a_1 and h_1 are the first row in A and H, respectively.

Keep W unpartitioned:

$$W = \begin{pmatrix} w_{11} & ... & w_{1m} \\ ... & ... & ... \\ w_{r1} & ... & w_{rm} \end{pmatrix}, \tag{2.37}$$

Using the partitioning idea, write WH as a "matrix-vector product":

$$WH = \begin{pmatrix} w_{11} & ... & w_{1m} \\ ... & ... & ... \\ w_{r1} & ... & w_{rm} \end{pmatrix} \begin{pmatrix} h_1 \\ ... \\ h_m \end{pmatrix} = \begin{pmatrix} w_{11}h_1 + ... + w_{1m}h_m \\ ... \\ w_{r1}h_1 + ... + w_{rm}h_m \end{pmatrix} \tag{2.38}$$

Look at that! What it says is row i of WH, and thus approximately row i of A, is a linear combination of the rows of H. And with a different partitioning, we'd find that each column of WH is a linear combination of the columns of H.

2.7 Random Vectors

Suppose we have random variables $X_1, X_2, ..., X_k$. We can form the vector

$$X = (X_1, ..., X_k)' \tag{2.39}$$

2.7.1 Expected Values and Covariance Matrices

The expected value is again a vector:

$$EX = (EX_1, ..., EX_k)' \tag{2.40}$$

Expected values are linear:

$$E(cX + dY) = cEX + dEY \tag{2.41}$$

for any constant scalars c and d, whether or not X and Y are independent.

The analog of variance is now the *covariance matrix*, $Cov(X)$, whose (i, j) element is $Cov(X_i, X_j)$. Thus the diagonal elements are the variances.

If the random vectors X and Y are independent,

$$Cov(cX + dY) = c^2 Cov(X) + d^2 Cov(Y) \tag{2.42}$$

for any constant scalars c and d.

For any conformable case of constant matrix A and random variable X,

$$Cov(AX) = A\, Cov(X)\, A' \tag{2.43}$$

An important special case is that in which A is a column vector c. Then

$$Var(c'X) = c'\, Cov(X)\, c \tag{2.44}$$

Note that the the covariance matrix here is 1×1, since $c'X$ is a scalar, so the matrix reduces to ordinary variance.

2.7.2 Diagonalization

Say we have a random vector X, $i \times 1$, with covariance matrix C. The latter is symmetric and real, so from (2.17), there is a matrix U such that

$$UCU' = D \tag{2.45}$$

where the columns of U are eigenvectors of C, and D is a diagonal matrix consisting of the eigenvalues of C. We say that U *diagonalizes* C.

But there is more. Define

$$Y = UX \tag{2.46}$$

Then, from (2.43),

$$Cov(Y) = UCU = D \tag{2.47}$$

In other words:

- We form new variables, the Y_j, from the original X_i, by multiplying X by the eigenvectors of C.
- These new variables, the Y_i, are uncorrelated (since the off-diagonal elements of D are 0s).
- $Var(Y_i) = \lambda_i$, the eigenvalues of C.

This will have major implications in the next chapter.

Chapter 3

Principal Components Analysis

3.1 The Notion of Approximate Rank

Suppose the matrix in (2.18) had been

$$M = \begin{pmatrix} 1 & 5 & 1 & -2 \\ 8.02 & 2.99 & 2 & 8.2 \\ 9 & 8 & 3 & 6 \end{pmatrix} \tag{3.1}$$

Intuitively, we still might say that the rank of M is "approximately" 2. Or better yet, row 3 still seems redundant, Let's formalize that, leading to one of the most common techniques in statistics/machine learning.

3.1.1 Dimension Reduction

One of the major themes in computer science is *scale*, as in the common question, "Does it scale?" The concern is, does an algorithm, method or whatever work well in large-scale systems?

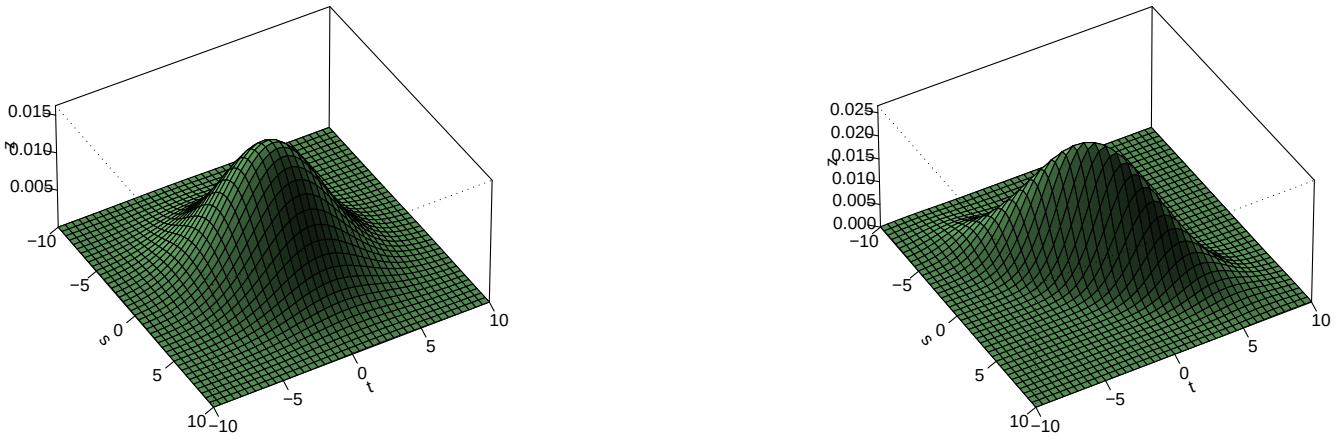
Just think of, say, Amazon. The business has millions of users and millions of items. In other words, one of their matrices might have millions of rows and millions of columns, and even one million rows and columns would mean a total number of $(10^6)^2 = 10^{12}$ entries, about 8 terabytes of data.

This is a core point in statistics/machine learning, the notion of *dimension reduction*. In complex applications, there is a pressing need to reduce the number of variables down to a manageable number — manageable not only in terms of computational time and space, but also the statistical problem of *overfitting*.

So we need methods to eliminate redundant or near-redundant data, such as row 3 in (3.1).

3.1.2 The Intuition

Statistically, the issue is one of correlation. In (3.1), the third row is highly correlated with (the sum of) the first two rows. To explore the correlation idea further, here are two graphs of bivariate normal densities:



Let's call the two variables X_1 and X_2, say human height and weight, with the corresponding axes in the graphs to be referred to as t_1 and t_2. The first graph was generated with a correlation of 0.2 between the two variables, while in the second one, the correlation is 0.8.

Not surprisingly due to the high correlation in the second graph, the "two-dimensional bell" is concentrated around a straight line, specifically the line $t_2 = -t_1$. In other words, there is high probability that $X_2 \approx -X_1$, so that X_2 is largely redundant. So:

> To a large extent, there is only one variable here, X_1 (or other choices, e.g. X_2), not two.

In the case of correlation 0.2, there really are two separate variables. The probability that $X_2 \approx -X_1$ is lower here.

Note one more time, though, the approximate nature of the approach we are developing. There really *are* two variables even in that correlation 0.8 example. By using only one of them, **we are relinquishing some information. But with the need to avoid overfitting, use of the approximation may be a net win for us.**

Well then, how can we determine a set of near-redundant variables, so that we can consider omitting them from our analysis? Let's look at those graphs a little more closely.

Any *level set* in the above graphs, i.e. a curve one obtains by slicing the bells parallel to the (t_1, t_2) plane can be shown to be an ellipse. As noted, the major axis of the ellipse will be the line $t_1 + t_2 = 0$. The minor axis will be the line perpendicular to that, $t_1 - t_2 = 0$. Now consider the variables

$$Y_1 = X_1 + X_2 \tag{3.2}$$

and

$$Y_2 = X_1 - X_2 \tag{3.3}$$

Suppose also that we have *scaled* the X_i, i.e. divided each by its standard deviation (common in data science more on this below), so each has variance 1.0.

Then, using (1.12), we have

$$Cov(Y_1, Y_2) = Var(X_1) - Var(X_2) + Cov(X_1, X_2) - Cov(X_1, X_2) = 0 \tag{3.4}$$

So Y_1 and Y_2 are uncorrelated, and since Y_2 is seen to not vary much, we have a good case for using only Y_1 in our data analysis, instead of using X_1 and X_2.

But why not use just X_1? As usual in statistics/ML, things get more complicated in higher dimensions. In choosing variables to retain in our analysis, it makes sense to require that they be uncorrelated, as Y_1 and Y_2 are above; if not, intuitively there is some redundancy among them, which of course is what we are hoping to avoid. Remember, we want to reduce the number of variables we'll work with, and redundancy would say that we might be able to reduce further.

With that in mind, now suppose we have p variables, $X_1, X_2, ..., X_p$, not just two. If our data is on people, these variables may be height, weight, age, gender and so on. We can no longer visualize in higher dimensions, but one can show that the level sets will be p-dimensional ellipsoids. These now have p axes rather than just two, and we can define p new variables, $Y_1, Y_2, ..., Y_p$ from the X_i, such that:

(a) The Y_i are uncorrelated.

(b) They are ordered in terms of variance:

$$Var(Y_1) > Var(Y_2) > ... > Var(Y_p) \tag{3.5}$$

Now we have a promising solution to our dimension reduction problem. In (b) above, we can choose to use just the first few of the Y_i, omitting the ones with small variance since they are essentially constants, uninformative. And again, since the Y_i will be uncorrelated, we are eliminating a source of possible redundancy among them.

PCA won't be a perfect solution — there is no such thing — as might be the case if the relations between variables is nonmonotonic. A common example is age, with mean income given age tending to be a quadratic (or higher degree) polynomial relation. But PCA is a very common "go to" method for dimension reduction, and may work well even in (mildly) nonmonotonic settings.

So, how do we find these Y_i?

3.1.3 Eigenanalysis

Say I have a sample data consisting of n observations on p variables, on n people. Let x_{ik} denote the data for Person k for variable i. Then, the *sample covariance matrix* C for this data has as its (i, j) element

$$\frac{1}{n}\sum_{k=1}^{n}(x_{ik} - \overline{x}_i)(x_{jk} - \overline{x}_j) \tag{3.6}$$

where $\overline{x}_i$ and $\overline{x}_j$ are the samples means of Variables i and j This is the sample-data analog of the covariance matrix, and all the properties in Section 2.7.2 hold. So we know how to get the Y_i! We compute the eigenvectors and eigenvalues of C, and proceed as in Section 2.7.2:

3.1.4 PCA

Typically we have many cases in our data, say n, arranged in an $n \times p$ matrix Q, with row i representing case i and column j representing the j^{th} variable.

Say our data is about people, 1000 of them, and we have data on height, weight, age, gender, years of schooling and income. Then $n = 1000$, $p = 6$.

So, finally, here is PCA:

1. Find the covariance matrix of the data, as in (3.6).
2. Compute its eigenvalues and eigenvectors. The eigenvalues are the variances of our new variables.
3. After ordering the eigenvalues from largest to smallest, let λ_i be the i^{th} largest, and let U_i be the corresponding eigenvector, scaled to have length 1.
4. Let U be the matrix whose i^{th} column is U_i. Its size will be $p \times p$.
5. Choose the first few eigenvalues, say s of them, using some criterion (see below). Denote the matrix of the first s columns of U by $U^{(s)}$.

6. Form a new data matrix,

$$R = QU^{(s)} \tag{3.7}$$

R will be of size $n \times s$. So, we've replaced our original p variables, the p columns of Q, by s new variables, the columns of R. Column j of R is called the j^{th} principal component of the original data.[1] is the reverse of that in (2.46, an algebraic effect of going from population distributions to sample values.)

As mentioned, the variance of the j^{th} principal component is λ_j. The sum of all p eigenvalues is the same as the sum of the variances of the original variables, an important point.

From this point onward, any data analysis we do will be with R, not Q. In R, row i is still data on the i^{th} case, e.g. the i^{th} person, but now with s new variables in place of the original p. Since typically $s << p$, we have achieved considerable dimension reduction.

3.1.5 Applying Matrix Partitioning

Using the approach of Section 2.6, partition $U^{(s)}$ into its columns,

$$U^{(s)} = (U_1, ..., U_s) \tag{3.8}$$

and thus write

$$R = QU^{(s)} = Q(U_1, ..., U_s) \tag{3.9}$$

After that second equality, pretend that Q and the U_i are "numbers." Then the last expression in (3.9),

$$Q(U_1, ..., U_s) \tag{3.10}$$

is a "scalar" times a "vector," and is thus equal to

$$(QU_1, ..., QU_s) \tag{3.11}$$

So, the i^{th} column of R is QU_i. The latter quantity is of the Ax, matrix-times-vector form of Section 2.6.3, so it is a linear combination of the columns of Q, with the coefficients in that linear combination being the elements in U_i.

[1] The ordering in (3.7)

Recall that each column of Q is one variable; e.g. for people, there may be an age column, a height column, a weight column and so on. Each column in R is one of our new variables. Therefore:

> The i^{th} new variable is equal to a linear combination of the old variables.

So, a new variable might be, say, 1.9 age + 0.3 height + 1.2 weight.

3.1.6 Choosing the Number of Principal Components

The number of components we use, s, is called a *tuning parameter* or *hyperparameter*. So, how do we choose s? This is the hard part, and there is no universal good method. Typically s is chosen so that

$$\sum_{j=1}^{s} \lambda_j \tag{3.12}$$

is "most" of total variance (again, that total is the above expression with p instead of s), but even this is usually done informally.

In many ML settings, though, s is chosen by cross validation.

3.1.7 Software and Example

The most commonly used R function for PCA is **prcomp()**. As with many R functions, it has many optional arguments; we'll take the default values here.

For our example, let's use the Turkish Teaching Evaluation data, available from the UC Irvine Machine Learning Data Repository. It consists of 5820 student evaluations of university instructors. Each student evaluation consists of answers to 28 questions, each calling for a rating of 1-5, plus some other variables we won't consider here.

```
> turk <- read.csv('turkiye-student-evaluation.csv',header=T)
> head(turk)
  instr class nb.repeat attendance difficulty Q1 Q2 Q3 Q4
1     1     2         1          0          4  3  3  3  3
2     1     2         1          1          3  3  3  3  3
3     1     2         1          2          4  5  5  5  5
4     1     2         1          1          3  3  3  3  3
5     1     2         1          0          1  1  1  1  1
6     1     2         1          3          3  4  4  4  4
```

```
   Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15 Q16 Q17 Q18 Q19
1   3  3  3  3  3   3   3   3   3   3   3   3   3   3   3
2   3  3  3  3  3   3   3   3   3   3   3   3   3   3   3
3   5  5  5  5  5   5   5   5   5   5   5   5   5   5   5
4   3  3  3  3  3   3   3   3   3   3   3   3   3   3   3
5   1  1  1  1  1   1   1   1   1   1   1   1   1   1   1
6   4  4  4  4  4   4   4   4   4   4   4   4   4   4   4
   Q20 Q21 Q22 Q23 Q24 Q25 Q26 Q27 Q28
1    3   3   3   3   3   3   3   3   3
2    3   3   3   3   3   3   3   3   3
3    5   5   5   5   5   5   5   5   5
4    3   3   3   3   3   3   3   3   3
5    1   1   1   1   1   1   1   1   1
6    4   4   4   4   4   4   4   4   4
> tpca <- prcomp(turk[,-(1:5)]
```

Let's explore the output. First, the standard deviations of the new variables:

```
> tpca$sdev
 [1] 6.1294752 1.4366581 0.8169210 0.7663429 0.6881709
 [6] 0.6528149 0.5776757 0.5460676 0.5270327 0.4827412
[11] 0.4776421 0.4714887 0.4449105 0.4364215 0.4327540
[16] 0.4236855 0.4182859 0.4053242 0.3937768 0.3895587
[21] 0.3707312 0.3674430 0.3618074 0.3527829 0.3379096
[26] 0.3312691 0.2979928 0.2888057
> tmp <- cumsum(tpca$sdev^2)
> tmp / tmp[28]
 [1] 0.8219815 0.8671382 0.8817389 0.8945877 0.9049489
 [6] 0.9142727 0.9215737 0.9280977 0.9341747 0.9392732
[11] 0.9442646 0.9491282 0.9534589 0.9576259 0.9617232
[16] 0.9656506 0.9694785 0.9730729 0.9764653 0.9797855
[21] 0.9827925 0.9857464 0.9886104 0.9913333 0.9938314
[26] 0.9962324 0.9981752 1.0000000
```

This is striking. The first principal component (PC) already accounts for 82% of the total variance among all 28 questions. The first five PCs cover over 90%. This suggests that the designer of the evaluation survey could have written a much more concise survey instrument with almost the same utility.

Now keep in mind that each PC here is essentially a "super-question" capturing student opinion via a weighted sum of the original 28 questions. Let's look at the first two PCs' weights:

```
> tpca$rotation[,1]
```

```
        Q1         Q2         Q3         Q4         Q5
-0.1787291 -0.1869604 -0.1821853 -0.1841701 -0.1902141
        Q6         Q7         Q8         Q9        Q10
-0.1870812 -0.1878324 -0.1867865 -0.1823915 -0.1923626
       Q11        Q12        Q13        Q14        Q15
-0.1866948 -0.1862382 -0.1922729 -0.1911814 -0.1902380
       Q16        Q17        Q18        Q19        Q20
-0.1962885 -0.1808833 -0.1935788 -0.1927359 -0.1931985
       Q21        Q22        Q23        Q24        Q25
-0.1911060 -0.1908591 -0.1948393 -0.1931334 -0.1888957
       Q26        Q27        Q28
-0.1908694 -0.1897555 -0.1886699


> tpca$rotation[,2]
        Q1         Q2         Q3         Q4         Q5
0.35645673 0.23223504 0.11551155 0.24533527 0.20717759
        Q6         Q7         Q8         Q9        Q10
0.20075314 0.24290761 0.24901577 0.12919618 0.18911720
       Q11        Q12        Q13        Q14        Q15
0.11051480 0.21203229 -0.10616030 -0.15629705 -0.15533847
       Q16        Q17        Q18        Q19        Q20
-0.04865706 -0.26259518 -0.12905840 -0.15363392 -0.19670071
       Q21        Q22        Q23        Q24        Q25
-0.22007368 -0.22347198 -0.10278122 -0.06210583 -0.20787213
       Q26        Q27        Q28
-0.12045026 -0.07204024 -0.21401477
```

The first PC turned out to place approximately equal weights on all 28 questions. The second PC, though, placed its heaviest weight on Q1, with substantially varying weights on the other questions.

While we are here, let's check that the columns of U are orthogonal.

```
> t(tpca$rotation[,1]) %*% tpca$rotation[,2]
             [,1]
[1,] -2.012279e-16
```

Yes, 0 (with roundoff error). As an exercise in matrix partitioning, the reader should run

```
t(tpca$rotation) %*% tpca$rotation
```

then check that it produces the identity matrix I, then ponder why this should be the case.

3.1.8 More on the PC Coefficients

There is more to consider.

Do the PC coefficients have any interpretation? The answer is probably no for ordinary people, but for the *domain experts*, very possibly yes. In the teaching evaluation example above, a specialist in survey design or teaching methods may well be able to interpret the dominance of Q1 in the second PC. A method called *factor analysis*, an extension of PCA, is popular in social science research.

For the rest of us, PCA is just a handy way to do dimension reduction.

But there is geometric terminology that will be helpful, as follows. Let's look at the **mlb** dataset from the **regtools** package. This is data on Major League baseball players.

```
                Name Team        Position Height Weight   Age
1    Adam_Donachie  BAL         Catcher     74    180 22.99
2        Paul_Bako  BAL         Catcher     74    215 34.69
3 Ramon_Hernandez  BAL         Catcher     72    210 30.78
4    Kevin_Millar  BAL   First_Baseman     72    210 35.43
5      Chris_Gomez  BAL   First_Baseman     73    188 35.71
6   Brian_Roberts  BAL  Second_Baseman     69    176 29.39
  PosCategory
1     Catcher
2     Catcher
3     Catcher
4  Infielder
5  Infielder
6  Infielder
```

Let's apply PCA:

```
> hw <- as.matrix(mlb[,4:5])
> pcout <- prcomp(hw)
> pcout$rotation
                PC1          PC2
Height -0.05948695  0.99822908
Weight -0.99822908 -0.05948695
```

If we were to plot **hw**, we would put **hw[1,]** at the point (74,180) on our graph. Recall from high school math that 74 and 180 are called the *coordinates* of **hw2[1,]**, with respect to our "H axis" and "W axis."

But in doing PCA, we are creating new axes, PC1 and PC2, which are rotated versions of the H and W axes. (Hence the naming of the U matrix as "rotation" in the **prcomp()** return value.) Let's find the coordinates of **hw[1,]** with respect to the new axes:

```
> hw[1,] %*% pcout$rotation
          PC1      PC2
[1,] -184.0833 63.1613
```

So (74,180) has become (-184.1,63.2) under the new coordinate system. Let's see what the angle of rotation is. We can do that by seeing where a point on the H axis rotates to.

```
> pc10 <- c(1,0) %*% pcout$rotation
> pc10
           PC1        PC2
[1,] -0.05948695 0.9982291
> (atan(pc10[2] / pc10[1])) * 180/pi
[1] -86.58964
```

Almost 90 degrees clockwise.

3.1.9 Scaling

Some analysts prefer to *scale* the data before applying PCA. For each column, we would subtract the column mean and divide by the column standard deviation. The column would now have mean 0.0 and variance 1.0.

The rationale for doing this is that if PCA is applied to the original data, variables with large variance will dominate. And then units would play a role; e.g. a distance variable would have more impact if it were measured in kilometers than miles.

Scaling does solve this problem, but its propriety is questionable. Consider a setting with two features, A and B, with variances 500 and 2, respectively, and with mean 100 for both. Let A' and B' denote these features after centering and scaling.

As noted, PCA is all about removing features with small variance, as they are essentially constant. If we work with A and B, we would of course use only A. But if we work with A' and B', we would use both of them, as they both have variance 1.0.

So, dealing with the disparate-variance problem (e.g. miles vs. kilometers) shouldn't generally be solved by ordinary scaling, i.e. by dividing by the standard deviation. An alternative is to divide each column by its mean. This addresses the miles-vs.-kilometers problem, and makes sense in that a variance is large or small in relation to its mean.

3.2 Vector Norms

In math, the l_p *norm* of a vector in n-dimensional space is defined by

$$||x||_p = \left(\sum_{i=1}^{n} [x_i|^p \right)^{1/p} \tag{3.13}$$

This is actually a family of norms, for $1 \leq p \leq \infty$.[2] You are familiar with the Euclidean norm, $p = 2$. In statistics and machine learning, we are often minimizing the norm of some vector, usually with either $p = 1$ or $p = 2$.

You will often see the notation L_p instead of l_p. However, in math the former is used for function spaces, while the latter designates n-dimensional vectors, and we use the latter here.

You will also see references to the *Frobenius* norm of an $r \times s$ matrix. That is actually just the l_2 norm, treating the matrix as an rs-dimensional vector.

3.3 Matrix Derivatives

There is an entire body of formulas for taking derivatives of matrix-valued expressions. One of particular importance to us is for the vector of derivatives

$$\frac{dg(s)}{ds} \tag{3.14}$$

for a vector s of length k. This is the *gradient* of $g(s)$, i.e. the vector

$$(\frac{\partial g(s)}{\partial s_1}, ..., \frac{\partial g(s)}{\partial s_k})' \tag{3.15}$$

A bit of calculus shows that the gradient can be represented compactly. in some cases, such as

$$\frac{d}{dt}(Mt + w) = M' \tag{3.16}$$

[2] $||x||_\infty = \max_i |x_i|$

where $t = (t_1, ..., t_k)'$, and M and w are $r \times k$ and $r \times 1$, and M and w are constants, i.e. do not depend on t. Of course, this is the matrix analog of the familiar

$$\frac{d}{dt}(ct + d) = c \tag{3.17}$$

from ordinary calculus.

The reader should verify (3.16) by looking at the individual $\frac{\partial g(s)}{\partial t_i}$.

Another example is the *quadratic form*

$$\frac{d}{dt}(t'Ht = 2Ht) \tag{3.18}$$

for a symmetric matrix H not depending on t, and a vector t. Again, it makes good intuitive sense for scalar t and H, where the relation would be

$$\frac{d}{dt}(Ht^2) = 2Ht \tag{3.19}$$

And there is a Chain Rule. For example if $t = Mv + w$, then

$$\frac{\partial}{\partial v}t't = 2M'v \tag{3.20}$$

3.3.1 Application to PCA

We can use matrix derivatives to develop an alternate derivation of PCA. For simplicity, we'll just find λ_1 here.

From our previous discussions, we know that finding the first principal component for a correlation matrix A amounts to finding a vector u such that

$$u'Au \tag{3.21}$$

is maximized. Of course, that by itself is not enough, as we could simply make u very large. Thus we need a constraint:

$$||u|| = u'u = 1 \tag{3.22}$$

One method of doing constrained optimization is that of *Lagrange multipliers*. Here we maximize

$$u'Au + \gamma(u'u - 1) \tag{3.23}$$

Here γ is an extra variable, i.e. we are maximizing with respect to both u and γ.

Using the above rules for taking matrix derivatives, we have

$$0 = \frac{d}{du}[u'Au + \gamma(u'u - 1)] = 2Au + \gamma 2u \tag{3.24}$$

and

$$0 = \frac{d}{d\gamma}[u'Au + \gamma(u'u - 1)] = u'u - 1 \tag{3.25}$$

Those two equations then give us

$$Au = -\gamma u \tag{3.26}$$

and

$$u'u = 1 \tag{3.27}$$

The second is our constraint, and the first says u is an eigenvector of A (with eigenvalue $-\gamma$)!