Linear Algebra

Most datasets that we'll be looking at are tables with columns corresponding to features and rows corresponding to examples. Suppose that the data that we're considering is **numeric** data. Then a given row/column can be identified with the mathematical structure called a vector, and the data itself can be viewed as a matrix. The study of vectors and matrices is subsumed by the subject of **Linear Algebra**. The subject figures very heavily in algorithms to analyze data.

Probably almost all of you have seen 2 and 3 dimensional vectors at some point in your education (for example, in high school physics). A general *vector* is just the analogous thing in n-dimensions. It can be visualized as an arrow (directed line segment) in n dimensional space. It can be represented as an array $\mathbf{v} = [v_1, v_2..., v_n]$ of real numbers, which specify the coordinates of the tip of the vector if the tail is placed at the origin.

Dot Product

The *dot product* of two vectors $\mathbf{v}$ and $\mathbf{w}$ is given by

$$\mathbf{v} \cdot \mathbf{w} = v_1 \cdot w_1 + v_2 \cdot w_2 + \cdots + v_n \cdot w_n$$

If $\mathbf{v}$ and $\mathbf{w}$ are zero centered, then $\mathbf{v} \cdot \mathbf{w}/n$ is just the *covariance* of $\mathbf{v}$ and $\mathbf{w}$. The *norm* of a vector $\mathbf{v}$ is defined by

$$||\mathbf{v}|| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

This is a generalization of the Euclidean distance (coming from the Pythagorean theorem) that you're familiar with.

We have

$$||\mathbf{v}||^2 = \mathbf{v} \cdot \mathbf{v} = v_1^2 + v_2^2 + \cdots + v_n^2$$

When $\mathbf{v}$ is zero-centered, $||\mathbf{v}||^2/n$ is the *variance* of $\mathbf{v}$, and $\sqrt{||\mathbf{v}||^2/n}$ i s the *standard deviation* of $\mathbf{v}$.

The cosine of the angle between the two vectors is given by

$$\cos\theta = \frac{\mathbf{v} \cdot \mathbf{w}}{||\mathbf{v}|| \cdot ||\mathbf{w}||}$$

This is also called the *cosine similarity*. When $\mathbf{v}$ and $\mathbf{w}$ are zero centered, this is the *correlation* between $\mathbf{v}$ and $\mathbf{w}$.

Linear Combinations

We frequently find ourselves adding vectors and multiplying them by numbers. If $\mathbf{v} = \{v_i\}$ and $\mathbf{w} = \{w_i\}$ then:

$$\mathbf{v} + \mathbf{w} = \begin{bmatrix} v_1 + w_1 \\ v_2 + w_2 \\ v_n + w_n \end{bmatrix}$$

and if a is a real number, then

$$a \cdot \mathbf{v} = \begin{bmatrix} a \cdot v_1 \\ a \cdot v_2 \\ \cdots \\ a \cdot v_n \end{bmatrix}$$

A *linear combination* of a collection of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$ is a vector of the form

$$a_1 \cdot \mathbf{v}_1 + a_2 \cdot \mathbf{v}_2 + \cdots + a_m \cdot \mathbf{v}_m$$

where the numbers a_i are real numbers.

In data science, we often find ourselves predicting a quantity using a linear combination of vectors. In a survey, 544 people were asked to report on their involvement in sports, watching TV, and watching TV sports on a scale from 1 to 10. Our data frame is

$$[\mathbf{sports}, \mathbf{tvsports}, \mathbf{tv}, \mathbf{gender}]$$

where "gender" is encoded as 1 for rows corresponding to males and 0 for rows corresponding to females. Using *linear regression*, which you'll be learning about tomorrow, we obtain the approximation

$$\mathbf{sports} \approx 4.8 + 0.78 \cdot \mathbf{gender} + 0.48 \cdot \mathbf{tvsports} - 0.18 \cdot \mathbf{tv}$$

which we can use use to predict how involved someone is in sports knowing their gender, how much they watch tv, and how much they watch tv sports.

Matrices

A matrix is an m by n grid of numbers like so:

$$M = \begin{bmatrix} 1 & 4 & 10 \\ 2 & 9 & 6 \\ 5 & 3 & 7 \\ 8 & 12 & 11 \end{bmatrix}$$

We can view it as an array of the columns (called *column vectors*) or an array of rows (called *row vectors*. The *transpose* of a matrix is the matrix obtained by interchanging the rows and the columns. We have

$$M^T = \begin{bmatrix} 1 & 2 & 5 & 8 \\ 4 & 9 & 3 & 12 \\ 10 & 6 & 7 & 11 \end{bmatrix}$$

Data frames can be represented as matrices. There are other important matrices that arise in data science as well. For example, the *covariance matrix* associated with a data frame is a matrix with the entry of the i^{th} row and j^{th} column being the covariance of feature i and feature j. In the example of the survey of involvement in activities, we get covariance matrix

$$COV = \begin{bmatrix} & sports & gender & tvsports & tv \\ sports & 6.92 & 0.33 & 3.61 & -0.49 \\ gender & 0.33 & 0.25 & 0.22 & -0.20 \\ tvsports & 3.61 & 0.22 & 7.85 & 1.72 \\ tv & -0.49 & -0.20 & 1.72 & 6.49 \end{bmatrix}$$

The correlation matrix is defined similarly: we get

$$COR = \begin{bmatrix} & sports & gender & tvsports & tv \\ sports & 1.00 & 0.25 & 0.49 & -0.07 \\ gender & 0.25 & 1.00 & 0.15 & -0.16 \\ tvsports & 0.49 & 0.15 & 1.00 & 0.24 \\ tv & -0.07 & -0.16 & 0.24 & 1.00 \end{bmatrix}$$

Note the symmetry about the diagonal: we have $COV = COV^T$ and $COR = COR^T$.

Product and Inverse

Given a matrix A and a vector $\mathbf{v}$, the product $A \cdot \mathbf{v}$ is defined to be the vector with i^{th} entry given by the dot product of the i^{th} row of A and $\mathbf{v}$. For example, if M is as above, and

$$\mathbf{v} = \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix}$$

then

$$A \cdot \mathbf{v} = \begin{bmatrix} 3 \cdot 1 + 2 \cdot 4 + 4 \cdot 10 \\ 3 \cdot 2 + 2 \cdot 9 + 4 \cdot 6 \\ 3 \cdot 5 + 2 \cdot 3 + 4 \cdot 7 \\ 3 \cdot 8 + 2 \cdot 12 + 4 \cdot 11 \end{bmatrix} = \begin{bmatrix} 51 \\ 48 \\ 49 \\ 92 \end{bmatrix}$$

If B is another matrix with number of rows equal to the number of columns of A, then we can form the product $A \cdot B$. This will be a matrix, with i^{th} row given by the product of A with the i^{th} column of B. Let $C = A \cdot B$, where A is an m by n matrix and B is an n by k matrix. Then C will be an m by k matrix.

When $m = 1$ and $k = 1$, A is a column vector, B is a row vector, and C is just a number giving the dot product of A and B.

Order of multiplication matters! In general, $AB \neq BA$.

When A is square (having equal numbers of rows and columns), it usually happens that there's a matrix A^{-1} with the property that $A \cdot A^{-1} = A \cdot A^{-1} = \mathbf{1}$, where $\mathbf{1}$ is a matrix with 1's on the diagonal and 0's elsewhere. If such a matrix exists, we call it the *inverse* of A, and say that A is *invertible.*

Eigenvectors and Eigenvalues

An especially simple kind of matrix is a *diagonal* matrix, with all 0's off of the main diagonal, such as:

$$A = \begin{bmatrix} 2 & 0 \\ 0 & -3 \end{bmatrix}$$

Given a vector $\mathbf{v}$, if we multiply it by A on the left. we get a new vector with first coordinate stretched by 2, and second coordinate stretched by 3 and reversed.

A square matrix A can often be *diagonalized*: there's often an invertible matrix B such that

$$B^{-1}AB = D$$

where D is diagonal, or equivalently

$$A = BDB^{-1}$$

This is equivalent to there being some coordinates along which multiplying by A from the left corresponds to stretching and/or reflection with respect to these (for example, a rotation of the standard coordinate system).

The stretching factors are called *eigenvectors* and the directions along which the stretching occurs are called *eigenvalues.* To find them, you find the values of λ for which $A\mathbf{v} = \lambda\mathbf{v}$ has a solution (to get the eigenvalues), and then for each eigenvalue, find the corresponding eigenvector. You won't need to do this by hand.

Numpy

The Python library numpy gives a good assortment of tools for doing manipulation with matrices and vectors.

You need to be careful though, because numpy's operations don't always match up exactly with what you would expect based on knowledge of linear algebra. For example, if we have a numpy array

$$A = \begin{bmatrix} 1 & 3 \\ 0 & 1 \end{bmatrix}$$

then you might expect `A * A` to be the matrix product of A with itself, which is

$$\begin{bmatrix} 1 & 6 \\ 0 & 1 \end{bmatrix}$$

but instead you get

$$\begin{bmatrix} 1 & 9 \\ 0 & 1 \end{bmatrix}$$

because numpy forms the matrix obtained by taking the product of respective entries to get an entry. Make sure that when you use numpy to take products, you use `np.dot`. If we type `np.dot(A, A)` into terminal, we do get A^2 as desired.