Linear Algebra and Python Applications

Klein
carlj.klein@gmail.com

1 Learning Objectives

These notes are to be used in reviewing the basics of linear algebra, and providing an analogue in programming the concepts in Python. We'll try to keep this relevant to the data science pathway, but some asides may be taken for pleasure in mathematics and necessity!

1. Matrix & Vector Basics
2. Dot Products & Cross Products
3. Matrix-Vector Products

2 Matrix & Vector Basics

We'll begin with the basics of linear algebra, which are rooted in a discussion about vectors and matrices.

- Operations on a Single Matrix
- Matrix Addition & Subtraction
- Matrix Multiplication (Dot Product)
- Vector Basics
- Standard Basis Vector (SBV)
- Span
- Linear Dependence & Independence
- Linear Subspaces
- Spans as Subspaces
- Basis

2.1 Operations on a Single Matrix

For somewhat of an analogue into data science and for review's sake, we could mention Gauss-Jordan elimination (or Gaussian elimination) to solve a system of equations. Essentially a series of row operations "mimicking" algebra. We'll skip the details and any coding in this section as the solutions available with the tools we'll build throughout the sections will be much more direct and less computationally expensive.

However, we should touch on number of solutions. Systems of equations can have a few different solutions:

- single, unique solution
- no solutions
- infinitely many solutions

2.2 Matrix Addition & Subtraction

For matrix addition and subtraction, matrices must be the same size, and then the operation is performed on corresponding entries between the matrices.
For example, say matrices A and B have addition or subtraction performed between them to form a matrix C. All matrices will have the same mxn (rows by columns) dimensions.

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix} \tag{1}$$

$$B = \begin{bmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \cdots & b_{m,n} \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix} \pm \begin{bmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \cdots & b_{m,n} \end{bmatrix} = \begin{bmatrix} c_{1,1} & \cdots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{m,1} & \cdots & c_{m,n} \end{bmatrix} \tag{3}$$

In other words...

$$A \pm B = C \implies a_{m,n} \pm b_{m,n} = c_{m,n} \; \forall m, n \tag{4}$$

For python, NumPy arrays will add correspondingly (element-wise), as is shown above. However, don't make the mistake of thinking lists, or lists of lists, will perform in this manner.

The following are always properties for Addition, but not always true for Subtraction:

- Commutative: A + B = B + A
- Associative: (A + B) + C = A + (B + C)

2.3 Matrix Multiplication (Dot Product)

Before diving into the dot product, we'll quickly mention scalar multiplication. Suppose we have a constant c and a Matrix A, then by performing scalar multiplication of A by c will multiply each element of A by c:

$$cA = \begin{bmatrix} c * a_{1,1} & \cdots & c * a_{1,n} \\ \vdots & \ddots & \vdots \\ c * a_{m,1} & \cdots & c * a_{m,n} \end{bmatrix} \tag{5}$$

$$cA \Longrightarrow c * a_{m,n} \; \forall m, n \tag{6}$$

Now to explain multiplying a matrix by another matrix. Dimensions and order of the matrices come into play here.

- Dimensions: the number of columns of the first matrix must be equal to the number of rows of the second matrix. The dimensions of the resulting matrix will be the number of rows of the first matrix by the number of columns of the second matrix.
- Order: A · B ≠ B · A

The dot product constructs a new matrix with each element being the sum of the first matrix's rows by the second matrix's columns.

For the dot product, the matrices can obviously be different dimensions as long as the requirements are met, so we'll slightly alter our generic matrices, A and B, with more specified labels.

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,n_a} \\ \vdots & \ddots & \vdots \\ a_{m_a,1} & \cdots & a_{m_a,n_a} \end{bmatrix} \tag{7}$$

$$\begin{bmatrix} b_{1,1} & \cdots & b_{1,n_b} \\ \vdots & \ddots & \vdots \\ b_{m_b,1} & \cdots & b_{m_b,n_b} \end{bmatrix} \tag{8}$$

Returning dimensions discussion for a dot product:

- $n_a = m_b$ must be true
- The resulting matrix will have dimensions m_a x n_b

An easy way to look at this is to examine the inner and outer dimensions of the matrices:

- $A_{dimensions} = m_a$ x n_a
- $B_{dimensions} = m_b$ x n_b
- Dimensions side by side: (m_a x n_a) (m_b x n_b)
- Required Dimensions: Inner Pair
- Resulting Dimensions: Outer Pair

The resulting matrix will be of the form:

$$\begin{bmatrix} \sum_{i=1}^{n_a} \sum_{j=1}^{m_b} a_{1,i} \cdot b_{j,1} & \cdots\cdots & \sum_{i=1}^{n_a} \sum_{j=1}^{m_b} a_{1,i} \cdot b_{j,n_b} \\ \vdots & \ddots\ddots & \vdots \\ \sum_{i=1}^{n_a} \sum_{j=1}^{m_b} a_{m_a,i} \cdot b_{j,1} & \cdots\cdots & \sum_{i=1}^{n_a} \sum_{j=1}^{m_b} a_{m_a,i} \cdot b_{j,n_b} \end{bmatrix} \tag{9}$$

Python requires a NumPy command, as using straight multiplication will multiply element-wise.

Here's an example of the dot product between two 2x2 matrices:

```
import numpy as np

A = np.array([[2, 6], [3, -1]])
B = np.array([[-4, -2], [1, 0]])
C = np.dot(A, B)

"""
array([[ -2,  -4],
       [-13,  -6]])
"""
```

Properties of matrix multiplication (dot product):

- NOT Commutative: AB $\neq$ BA
- Associative: (AB)C = A(BC)
- Distributive: A(B $\pm$ C) = AB $\pm$ AC
- Zero Matrix: Any matrix multiplied by the zero matrix will return a zero matrix, but depending on order, the dimensions could change

2.4 Vector Basics

A vector can be thought of as a special type of matrix in which one of it's dimensions is 1. It can be represented as a column vector (column dimension = 1), or a row vector (row dimension = 1). We can break down or describe any matrix in terms of its vectors, via rows or columns. Similarly, we can build a

matrix out of a collection of same length vectors.

Since a vector is a special case of a matrix, as long as dimensional requirements are met, algebraic operations (addition, subtraction, matrix multiplication) between other vectors and matrices is possible.

Vectors contain two pieces of information:

- Direction
- Magnitude (length)

Magnitude is simply the length of a vector, which can be calculated using the distance formula extended to the length of the vector.

Given a vector, v, with dimensions n:

$$\vec{v} = \begin{bmatrix} v_1 \\ \vdots \\ \vdots \\ \vdots \\ v_n \end{bmatrix} \tag{10}$$

We can calculate the magnitude as follows:

$$\|\vec{v}\| = \sqrt{v_1^2 + ... + v_n^2} \tag{11}$$

Given we now know how to find the magnitude of a vector, we can introduce the concept of the unit vector. Every vector has a vector of magnitude 1 which points in the same direction as it, this is the unit vector.

$$\vec{u} = \frac{1}{\|\vec{v}\|} \cdot \vec{v} \tag{12}$$

Syntactically, $\vec{u} = \hat{u}$, which represents a vector has length 1.
Calculating an example in python introduces another NumPy command (and sub-module). Introducing numpy.linalg and np.linalg.norm:

```
v = [1, 2, 3]
v_magnitude = np.linalg.norm(v)
# 3.7416573867739413
u = v / v_magnitude
# [0.26726124 0.53452248 0.80178373]
u_magnitude = np.linalg.norm(u)
# 1.0
```

2.5 Standard Basis Vector (SBV)

The standard basis vectors are a set of vectors which can build every vector in their own respective n-dimensional space by scaling each one. These scaled

combinations are called linear combinations.

Set of standard basis vectors in $\mathbb{R}^n$:

$$S = s_1, s_2, ..., s_n = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}, ... , \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ 1 \end{bmatrix} \tag{13}$$

There are n vectors, each of which has dimension n.

A linear combination can build any vector in their respective n-dimensional space:

$$\begin{bmatrix} v_1 \\ \vdots \\ \vdots \\ v_n \end{bmatrix} = \vec{v} = v_1 \cdot s_1 + ... + v_n \cdot s_n \tag{14}$$

2.6 Span

2.7 Linear Dependence & Independence

2.8 Linear Subspaces

2.9 Spans as Subspaces

2.10 Basis

3 Dot Products & Cross Products

4 Matrix-Vector Products