Vectors and their uses

Sharon Goldwater
Institute for Language, Cognition and Computation
School of Informatics, University of Edinburgh

DRAFT Version 0.95: 3 Sep 2015. **Do not redistribute without permission.**

This tutorial reviews the basics of vector arithmetic and gives a taste of some uses of vectors in cognitive science, natural language processing (NLP), and maching learning. If you've never seen vector arithmetic before and find the explanations here insufficient, there are some good videos from Khan Academy that should give you a lot more intuition about the relationship between the different ways of thinking about vectors, and what arithmetic operations on vectors mean (though note that Khan writes vectors as columns rather than rows as we do here).

If you want to watch the videos, the ones that cover roughly the material in this tutorial (except Sections 4 and 6) are the first five videos (up to 'Scaling Vectors') that start here:
`https://www.khanacademy.org/math/linear-algebra/vectors_and_spaces/vectors/v/vector-introduction-linear-algebra`
plus one additional video here:
`https://www.khanacademy.org/math/linear-algebra/vectors_and_spaces/dot_cross_products/v/vector-dot-product-and-vector-length`.

1 Definition, notation, dimensionality

You may have learned that a VECTOR is a quantity with both magnitude and direction. While this is one way of defining a vector, here we follow a different (though compatible, as we'll see below) definition. We say that a vector is simply an ordered sequence of numbers, normally denoted using square or round brackets, as in $[1.0, 2.3, -5.6]$ or $(0, 2)$.[1]

VECTOR

The number of items in the vector is also called its DIMENSIONALITY, for reasons that will become clear. So $[1.0, 2.3, -5.6]$ is a 3-dimensional vector and $(0, 2)$ is a 2-dimensional vector.

DIMENSIONALITY

If you've seen vectors before, you may have used special notation for the variables that refer to them, to distinguish from variables that refer to SCALARS (single numbers). For example, some people use an arrow or bar over a variable name to indicate that it refers to a vector, as in

SCALARS

$$\vec{x} = (1.0, 2.3, -5.6) \quad (1)$$

or

$$\bar{x} = (1.0, 2.3, -5.6) \quad (2)$$

These are good for handwritten maths, but in computer science textbooks and articles it is also common to see vectors denoted using bold variables, as in

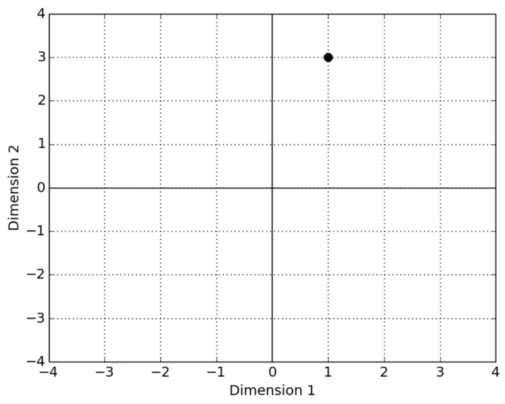
$$\mathbf{x} = (1.0, 2.3, -5.6) \quad (3)$$

[1] In linear algebra and parts of machine learning that make heavy use of linear algebra, you will also see *column vectors* in which the numbers are stacked vertically, as in $\begin{bmatrix} 0 \\ 2 \end{bmatrix}$. Here, we will just use row vectors.

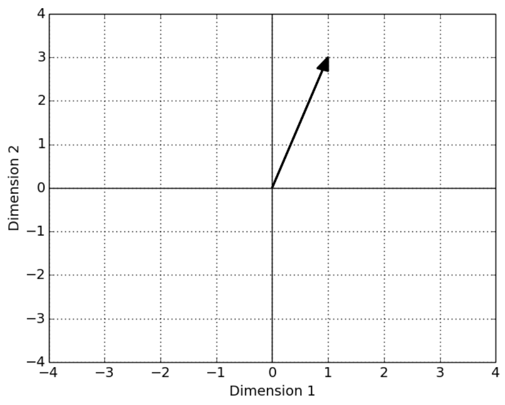
All of these are simply conventions, and you will also sometimes see vectors referred to using variables that don't have any special "vector" notations.

2 Vectors as points in multidimensional space, vector length

A d-dimensional vector can be interpreted as a point in d-dimensional space, where the i'th number in the vector corresponds to the coordinate along the i-th dimension in the space. For example, if $\mathbf{y} = (1,3)$, we can locate $\mathbf{y}$ on a 2-dimensional plot at coordinates $(1,3)$:



To recover the "magnitude and direction" view of vectors, we simply draw an arrow from the origin $(0,0)$ to the point $\mathbf{y}$:



This arrow shows graphically both the MAGNITUDE (more commonly called LENGTH or NORM) and direction of $\mathbf{y}$. Since this arrow representing $\mathbf{y}$ is the hypoteneuse of a triangle with sides of length 1 and 3, its length is $\sqrt{1^2+3^2}$ (computed using the Pythagorean formula). More generally, the length of a d-dimensional vector $\mathbf{z} = (z_1, z_2, \ldots z_d)$ is written as $\|\mathbf{z}\|$ (or sometimes just $|\mathbf{z}|$) and is computed as:

MAGNITUDE
LENGTH
NORM

$$\|\mathbf{z}\| = \sqrt{\sum_{i=1}^{d} z_i^2} \tag{4}$$

$$= \sqrt{z_1^2 + z_2^2 + \ldots z_d^2} \tag{5}$$

The summation notation $\sum$ in (4) is simply a shorthand for what is spelled out in (5); if you are uncomfortable with this notation please go through the Sums, Products, and Functions tutorial.

3 Adding and subtracting vectors, multiplying by a scalar

Suppose $\mathbf{x}$ and $\mathbf{y}$ are two d-dimensional vectors, so $\mathbf{x} = (x_1, x_2, \ldots x_d)$ and $\mathbf{y} = (y_1, y_2, \ldots y_d)$. To compute $\mathbf{x} + \mathbf{y}$, we simply add each dimension separately:

$$\mathbf{x} + \mathbf{y} = (x_1 + y_1,\ x_2 + y_2,\ \ldots x_d + y_d) \tag{6}$$

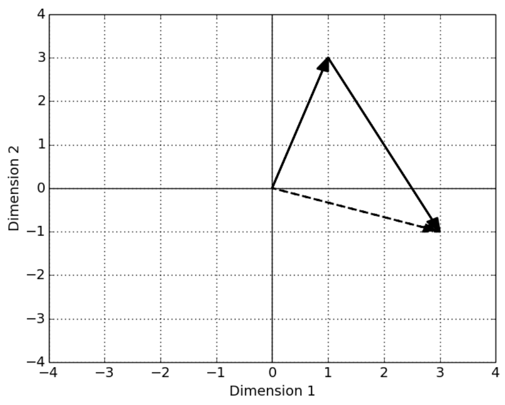
so $(1,3,6) + (6,2,8) = (7,5,14)$.

Not surprisingly, we do the same for subtraction:

$$\mathbf{x} - \mathbf{y} = (x_1 - y_1,\ x_2 - y_2,\ \ldots x_d - y_d) \tag{7}$$

so $(1,3,6) - (6,2,8) = (-5,1,-2)$.

Just as individual vectors can be interpreted geometrically, the addition of two vectors can also be interpreted geometrically. Again, for simplicity, we will use two-dimensional vectors. Let's start with the same vector we used in our earlier example: $\mathbf{y} = (1,3)$ and add to it the vector $\mathbf{x} = (2,-4)$. The answer we get from computing $\mathbf{y} + \mathbf{x}$ using Eq 6 is $(3,-1)$. This is represented as the dotted arrow in the figure, with its head at $(3,-1)$:



But notice that $(3,-1)$ is also what we get by placing the arrow representing the magnitude and direction of $\mathbf{y}$ with its tail at the origin (as we did earlier), and then placing the arrow representing the magnitude and direction of $\mathbf{x}$ *with its tail at the head of* $\mathbf{y}$. So, vector addition can also be viewed as following the magnitude and direction of each of the summed vectors in turn.

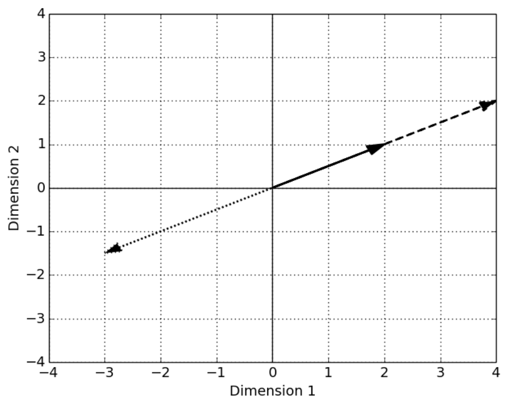
In this case, we placed the arrows in such a way as to represent $\mathbf{y} + \mathbf{x}$. You should convince yourself now that we would end up in the same location on the plot if we had instead added $\mathbf{x} + \mathbf{y}$. As with scalar addition, vector addition is COMMUTATIVE: order doesn't matter. COMMUTATIVE

Warning: Vectors can only be added together if they have the same dimensionality! Trying to add vectors of different dimensionality is not a legal or sensible operation.

Finally, we can multiply a vector $\mathbf{x}$ by a scalar a, which yields a vector with the same direction as $\mathbf{x}$ but a magnitude which is a times as big. We do so by multiplying each element separately:

$$a\mathbf{x} = (ax_1, ax_2, \ldots ax_d) \tag{8}$$

For example, if $\mathbf{x} = (2,1)$ (solid arrow), then $2\mathbf{x} = (4,2)$ (dashed arrow, which starts at origin but is partly covered by the solid arrow) and $-1.5\mathbf{x} = (-3,-1.5)$ (dotted arrow):



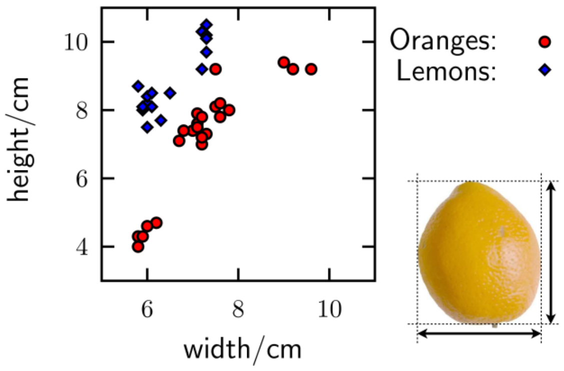
4 Representing data using vectors

In machine learning, NLP, and cognitive science, one of the most common uses of vectors is as a way of *representing information*. For example, each of the numbers in a vector might correspond to the value of some particular FEATURE of an object. So, maybe we have a bowl of fruit, and we measure the width and height of each piece of fruit. We could then represent each piece of fruit as a two-dimensional FEATURE VECTOR, in this case a (width, height) vector.

FEATURE

FEATURE VECTOR

Suppose there are a few different types of fruit—say, different varieties of oranges and lemons.[2] We also have a few different instances of each type. If we plot each of our (width, height) measurements as a separate data point, we might find that the data points (each representing a single piece of fruit) cluster into somewhat distinct groups, where each group is a particular variety of orange or lemon:



In the figure, oranges and lemons are given different colors/shapes, but notice that even within the oranges, there are a few distinct clusters; and similarly for the lemons. It turns out that those clusters roughly coincide with the different varieties of oranges and lemons.

The example with oranges and lemons is very simple, but it illustrates an important point. Different categories (here, varieties of fruit) often have different sets of properties (features). By representing those properties as vectors of feature values, we may be able to distinguish between different categories simply by looking at how the data points cluster together in VECTOR SPACE.

VECTOR SPACE

Of course, real problems are usually much more complex, and real data often has a much larger number of features. Sometimes we don't even know which features are relevant for the categorization task we are working on. I won't discuss that issue further here, but I will give

[2]This example, and the figure associated with it, are from the "oranges and lemons" data set here: `http://homepages.inf.ed.ac.uk/imurray2/teaching/oranges_and_lemons/`, which has a more extensive discussion of clustering and analysis of the data than I have here.

a slightly more realistic example to illustrate the use of feature vectors. This example deals with the task of DOCUMENT CLUSTERING: given a set of documents, try to cluster them into groups according to topic, for example politics or sport or travel.

DOCUMENT CLUSTERING

For this task, each document can be represented as a single vector, where the features in the vector are word counts. That is, each feature (dimension of the vector) represents a particular word in the vocabulary, and the value of that feature is the number of times the word appears in the document.

I'll start with a simplified example, where the vocabulary is only ten words. Our document vectors will represent (in order) the counts of each word:

```
the a and big small chased walked girl dog cat
```

Suppose we have two very short documents:

doc_1 :	`the big cat chased the big girl and the dog`
doc_2 :	`a small girl walked a big dog`

Then our vector representation of these documents would be:

$$\mathbf{v} = (3,0,1,2,0,1,0,1,1,1)$$
$$\mathbf{w} = (0,2,0,1,1,0,1,1,1,0) \tag{9}$$

But now what? Remember, we are trying to find groups of documents that share a topic. Intuitively, documents with a similar topic should use similar words, and therefore have similar feature vectors. Just like the oranges and lemons, documents that have similar feature vectors should be *close together* in the vector space we defined. So, what we need is a way to measure *distance* between vectors. The next section discusses a few standard ways to do so.

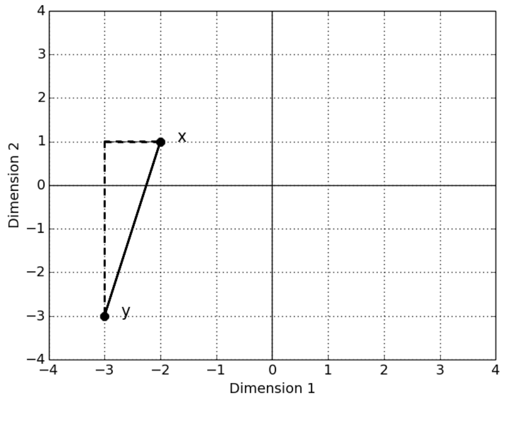
5 Euclidean distance, dot product, cosine similarity

In the preceding sections, we saw that vectors can be interpreted geometrically as points in a multi-dimensional space. So perhaps the most obvious way to measure the distance between two vectors **x** and **y** is to consider the length of a straight line connecting the two points representing **x** and **y**. This measure is known as the EUCLIDEAN DISTANCE, and is defined as

EUCLIDEAN DISTANCE

$$\text{EucDist}(\mathbf{x},\mathbf{y}) \quad = \quad \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2} \tag{10}$$

To see where this equation comes from, let's consider a two-dimensional example. Let $\mathbf{x} = (-2,1)$ and $\mathbf{y} = (-3,-3)$. In this case, we can compute the distance between the two points by constructing a right triangle whose legs are parallel to the axes of our space, and whose hypotenuse is the distance we want to compute:



From the picture, we see that the lengths of the legs are $|x_1 - y_1|$ and $|x_2 - y_2|$, so we use the Pythagorean theorem to compute the length of the hypotenuse as $\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} = \sqrt{1+16} = \sqrt{17}$.

The formula in (10) is simply a generalization of the two-dimensional case into multiple dimensions. Based on our intuitions from two- and three-dimensional spaces, it would seem to be a reasonable way to measure distance in a vector space. Unfortunately, our intuitions from two- and three-dimensional spaces can be misleading when we are working with higher dimensional spaces, and it turns out that Euclidean distance is often *not* the best way to measure distances in high-dimensional space.[3].

There are a lot of different alternative ways to measure distances, but I will just give one commonly used one here. Before we define this metric, we first need to define the DOT PRODUCT between two vectors **x** and **y**, written as $\mathbf{x} \cdot \mathbf{y}$:[4]

DOT PRODUCT

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{d} x_i y_i \tag{11}$$

That is, to get the dot product, we multiply together the values in each dimension in turn, and add up all the results. For our example document vectors in (9), we'd get:

$$\begin{aligned} \mathbf{v} \cdot \mathbf{w} &= 3\cdot 0 + 0\cdot 2 + 1\cdot 0 + 2\cdot 1 + 0\cdot 1 + 1\cdot 0 + 0\cdot 1 + 1\cdot 1 + 1\cdot 1 + 1\cdot 0 \\ &= 4 \end{aligned} \tag{12}$$

(where the dots on the righthand side indicate multiplication as you've always known it. As with much notation in maths, you need to understand from context whether the dot is a dot product of vectors, or just multiplication of two scalars.)

It should be clear that if x_i and y_i are both large (for text clustering, if both documents have many occurrences of a particular word), then $x_i y_i$ will be large, and contribute to a larger dot product. Whereas if x_i or y_i is small, then $x_i y_i$ will be relatively small, and will contribute less to the dot product. So, the dot product between two documents will be largest if those two documents contain a lot of words in common, and those shared words occur frequently in both documents. Very roughly, then, vector pairs with large dot products indicate document pairs with greater topic similarity, which is what we wanted from a similarity measure.

There are two problems with using the dot product as a similarity measure, however. First, you might have noticed that in our example, the *absence* of a particular word from both documents, i.e., having both x_i and y_i be small or zero, does not make any difference to the dot product. But coinciding absences like this might actually be informative about the similarity of documents. It turns out that for the particular problem of document clustering, one of the best ways to deal with this issue is not by changing the distance (or similarity) measure, but by changing the way the features are computed, so they aren't just word counts. However, I won't go into that here; it's better treated in an NLP or machine learning course.

The other problem is more straightforward to fix. Consider that documents may contain different numbers of words. A long document will tend to have larger word counts *in general* than a short document, simply because of its length. Put another way, a long document will be represented by a long vector: one with a large magnitude. But a vector with larger word counts overall will tend to look more similar to *all* other vectors, if dot product is our measure of similarity.

As a simple example, consider a new document that just contains two copies of doc_2, so its feature vector is $2\mathbf{w}$. The dot product of this new document with $\mathbf{v}$ is $2(\mathbf{v} \cdot \mathbf{w})$ (if you don't see why, write out the maths!). So, we got a bigger dot product just by making a longer document that repeats the same words more often.

To get a more sensible measure of similarity that doesn't depend on the document length (or, more generally, on the magnitude of our feature vectors), we need to NORMALIZE each

NORMALIZE

[3]StackExchange has a good discussion of this issue with a number of useful and interesting links: `http://stats.stackexchange.com/questions/99171/why-is-euclidean-distance-not-a-good-metric-in-high-dimensions`

[4]As usual, there are other notations too: you may see $\langle x, y \rangle$ or, if using column vectors, $x^T y$.

vector. That is, we divide each vector by its length so that all vectors end up with length 1 (UNIT LENGTH): they simply point to different locations on the surface of a sphere with radius 1 that is centered at the origin.

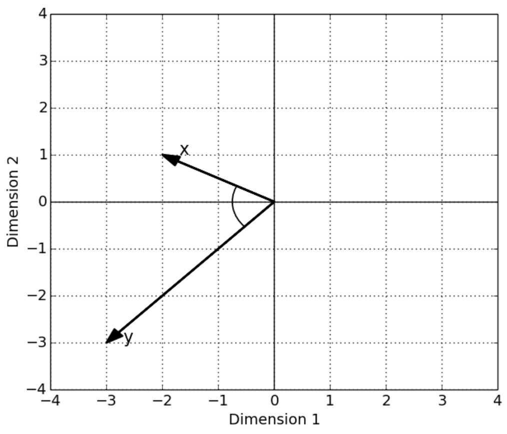
UNIT LENGTH

We already showed how to compute the length of the vector in Eq 4, so we use that equation here to define the NORMALIZED DOT PRODUCT, which is also called the COSINE SIMILARITY:

NORMALIZED DOT PRODUCT
COSINE SIMILARITY

$$\begin{aligned} \text{CosSim}(\mathbf{x},\mathbf{y}) &= \frac{\mathbf{x}}{||\mathbf{x}||}\cdot\frac{\mathbf{y}}{||\mathbf{y}||} \\ &= \frac{\mathbf{x}\cdot\mathbf{y}}{||\mathbf{x}||\,||\mathbf{y}||} \end{aligned} \tag{13}$$

The reason this metric is called cosine similarity is because it turns out that the value computed by Eq 13 is actually the cosine of the angle between **x** and **y**. So, for $\mathbf{x} = (-2,1)$ and $\mathbf{y} = (-3,-3)$ (the same vectors we used in the Euclidean distance example), Eq 13 gives the cosine of the angle shown below:



Recall that the cosine ranges from 1 (for an angle of 0°) to -1 (for an angle of 180°). So, two vectors that are coincidental will have a cosine similarity of 1; two vectors that are pointing in opposite directions will have a cosine similarity of -1; and two vectors that are orthogonal (at 90°) will have a cosine similarity of 0.

When just considering two-dimensional vectors, it isn't obvious why this metric is better than Euclidean distance, but for (some) high-dimensional data it can work better for clustering and other machine learning applications. In addition, the concept of normalizing vectors to unit length comes up in other contexts as well.

6 Interpretability of feature vectors, dimensionality reduction

As a final note, it's worth pointing out that the different dimensions used to represent data in multidimensional space are not always easily interpretable. In the examples above, I said that the dimensions were used to represent width and height (for fruit) or number of word occurrences (for documents). However, many modern machine learning methods take interpretable vectors like these, which often have very high dimensionality,[5] and perform some mathematical tricks to greatly *reduce* the dimensionality while preserving, as much as possible, the spatial relationships between the data points. In other words, points that are far apart in the original space should be far apart in the reduced dimensional space, and those that are close together in the original space should be close together in the reduced space. The resulting vectors can represent nearly the same information as the original ones while

[5] If we are dealing with language, the natural dimensionality is the number of words in the vocabulary, which can easily reach thousands or tens of thousands.

potentially having a number of benefits (such as requiring less computer memory to store, and being less sensitive to features that are noisy or irrelevant to the problem). The downside is that they are often less easy to interpret than the original vectors because the dimensions no longer correspond neatly to obvious features of the data. I won't go into dimensionality reduction methods here—they are taught in most machine learning courses—but it's good to be aware that they exist.

If you are likely to be taking a course that covers dimensionality reduction, or want to understand it better on your own, it's very useful to have some background in linear algebra. I have not looked in any detail at the content of the following linear algebra courses, but they are both available for free online and are likely to cover what you will need (if not more):

- Khan Academy course on linear algebra:
  `https://www.khanacademy.org/math/linear-algebra`
- Gilbert Strang's linear algebra course on MIT OpenCourseWare:
  `http://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/`

7 Exercises

1. For each vector, what is its dimension? What is its length?
   (a) $(3,4)$
   (b) $(2,-1,4,0)$
2. Compute $||\mathbf{x}||$ in each case.
   (a) $\mathbf{x} = (7,3,-0.1)$
   (b) $\mathbf{x} = (2,-3,1,2)$
3. What is the length of an n-dimensional vector with each dimension equal to 1?
4. Given $\mathbf{x} = (7,3,-0.1)$, compute the following:
   (a) $\mathbf{x} + (-1,0.3,2)$
   (b) $3\mathbf{x}$
   (c) $\frac{\mathbf{x}}{2} - (0.5,-2,0)$
5. For each pair of vectors $\mathbf{x}$ and $\mathbf{y}$ below, can you say whether the magnitude of $\mathbf{x}$ is greater than, equal to, or less than the magnitude of $\mathbf{y}$, *without* actually computing the magnitudes? If so, explain why. If not, compute the magnitudes to compare them.

   This exercise is intended to build *intuitions* about vectors, not just brute force calculation. In mathematics, intuitions are often more important than just memorizing an equation—we want to understand the implications of that equation. In this particular case, understanding the equation for computing vector length should allow you to determine the relative lengths in many of the pairs below without actually computing them.

   (a) $\mathbf{x} = (3,4,7)$, $\mathbf{y} = (-3,4,-7)$
   (b) $\mathbf{x} = (3,4)$, $\mathbf{y} = (2,-1,4,0,2)$
   (c) $\mathbf{x} = (2,1,-5,0)$, $\mathbf{y} = (0,2,-5,1)$
   (d) $\mathbf{x} = (2,1,-5,0)$, $\mathbf{y} = 2\mathbf{x}$
   (e) $\mathbf{x} = (2,1,3)$, $\mathbf{y} = (2,-5,7)$