Applications to machine learning

Machine learning is the study of how to make programs adapt in response to data. One particular kind of ml program is the classifier - given many examples of things that belong in one of several categories and their correct labels, the program should create a function that returns the correct category when presented with a new example.

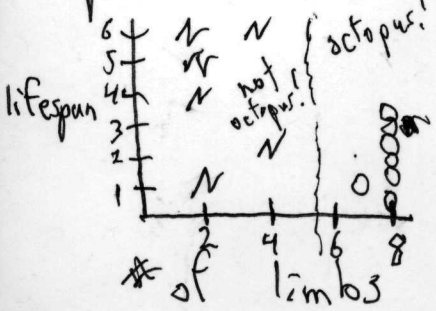
Example: Digit recognition. People feed the MNIST database of handwritten digits 0-9 to their programs, and the programs can then identify new images of digits as 0-9. Automatic digit recognition is used by USPS.

Example: Face detection & recognition. Rectangles in images are classified as containing faces or not. Once we have a face, we could decide which of several people it's likely to be.

In many cases the training examples and test inputs for classifiers can be seen as vectors.

- A grayscale image with k pixels can be thought of as a k-dimensional vector of values in $[0, 255]$ corresponding to the brightness of each pixel.
- A sound wave can be decomposed into a vector of frequencies and their amplitudes.
- Numerical demographic data such as age, height, weight can become components of a vector. Boolean values can become 0 or 1.

The data points used to train a classifier can be seen as vectors or points in a k-dimensional space, where k is the number of features (components) per vector.



[octopus classifier]

The goal of the classifier is then to return the most promising classification given the location of a data point in the space. We don't have nearly enough data points to fill the space, but we'll try to come up with a hypothesis that reasonably fits the data without being too complex.

There are generally an infinite number of hypotheses that are consistent with the existing data, but classifiers generally work better if they don't make things more complex than necessary.

A <u>linear classifier</u> doesn't have the power to represent overly complex hypotheses. It will always decide on a separating surface that is a straight line, plane, or hyperplane.

The <u>perceptron</u>, an early neural network design, is an example of a linear classifier. Its final classifying function is of the form

if $a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \geq b$

return <u>true</u>

else

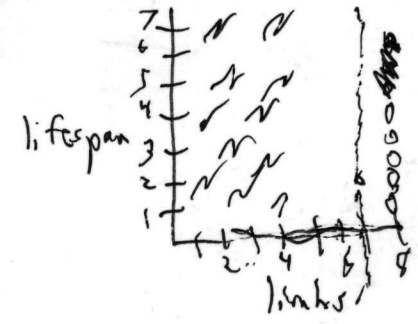
return <u>false</u>

(x_n: input; a_n: classifier parameter (learned))

Notice that $a_1 x_1 + a_2 x_2 + \dots + a_n x_n = b$ is the equation of a hyperplane, $> b$ is one side of the hyperplane, and $< b$ is the other.

Our octopus perceptron could use parameters $a_1 = 1$, $a_2 = 0$, $b = 7$ and have the hypothesis

```
if 1 * limbs + 0 * lifespan ≥ 7
        return TRUE
else
        return FALSE
```



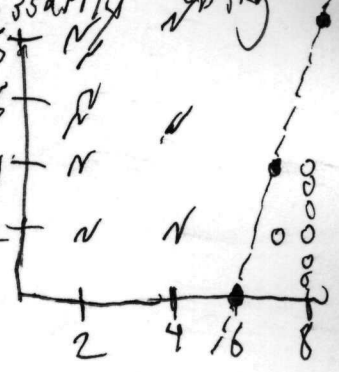
Or if we wanted to better fit that outlier,

```
if 4 * limbs − lifespan ≥ 24
        return TRUE
else
        return FALSE
```

$y = 4x - 24$

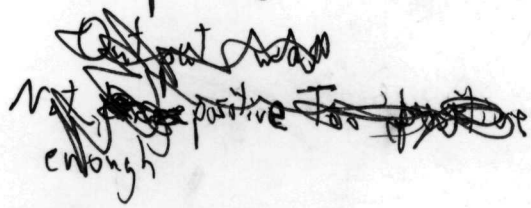
$4x - y = 24$

which doesn't make a lot of sense, but does produce a cleaner separation in the space. In practical situations, it can be good to remember that classifiers aren't necessarily using a common sense understanding of the world. What it *is* trying to do (in the case of a linear classifier) is weight all the factors according to the strength and direction of their effect on the classification.



So how do we program something that will automatically adapt this function to the data? Many ML methods work by trying a hypothesis, determining how wrong they were, then nudging the hypothesis in a direction that would have been more correct. This is the basis of the perceptron learning rule.

The goal of the perceptron learning rule is to adjust each parameter of the learner in a direction more likely to produce correct output next time.



Input was...	Output needs to be... Reduced	Increased
+	Reduce weight	Increase weight
−	Increase weight	Decrease weight

Let the error be (desired − output), where output is 1 if $a_1x_1 + \dots + a_nx_n \geq b$ and 0 otherwise, and desired is 1 if we should return true and 0 otherwise. So

$$\text{error} = \begin{cases} 0 & \text{if output} = \text{desired} \\ -1 & \text{if we said "yes" when we shouldn't} \\ 1 & \text{if we said "no" when we shouldn't.} \end{cases}$$

The error is in the direction we should adjust.

The perceptron learning rule is

$$w_i' = w_i + \alpha \cdot \text{err} \cdot x_i$$

where α is a "learning rate" $0 < \alpha \leq 1$ that governs the "step size" or amount of adjustment. (Without this, the hypothesis might overshoot a good answer forever.)

We can verify that the direction of adjustment matches what we want.

Input was...	Output needs to be... Reduced (err=−1)	Increased (err=1)	same (err=0)
+	Decrease weight	Increase weight	no change
−	Increase weight	Decrease weight	no change

The "x_i" causes the adjustment to be opposed to the error.

The "x_i" flips the adjustment if x_i was negative. Including x_i The magnitude of x_i reflects the fact that inputs of bigger magnitude have bigger effects on output.

The threshold we're comparing against could be lowered if our error was to say "no" (1) and raised if we mistakenly said yes (-1). But to have a single consistent rule, perceptrons sometimes assume the threshold is 0 but include a bias term b:

$a_1 x_1 + a_2 x_2 + \dots + a_n x_n + b \geq 0 \iff a_1 x_1 + \dots + a_n x_n \geq -b$

We can think of b as just another weight of an input that is always 1: $b = b(1)$. Then the normal perceptron learning rule tells us

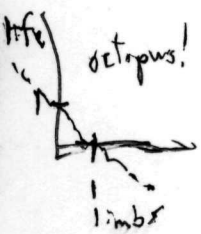
$b' = b + \alpha \, err(1) = b + \alpha \, err.$

We can't throw out the bias altogether — without any offset the zero vector is always a solution to $a_1 x_1 + \dots a_n x_n \geq 0$ and that would force the decision boundary to pass through zero. As we see with the octopus classifier that may not make sense.

The perceptron learning algorithm then starts with random weights in the classifier and iterates on the training data until there is a classifier that gets everything right. If such a solution exists, it will be found. If it doesn't, the algorithm will just run until we give up.

Example: Octopus classifier. Arbitrary starting weights $\alpha = 0.5$.

$x + y - 1 \geq 0$.



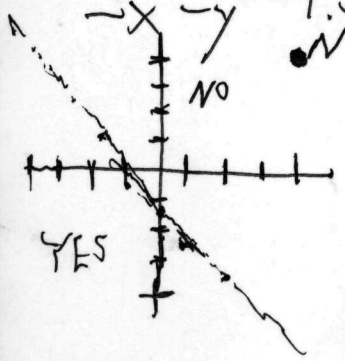
Get a datapoint $\begin{bmatrix} 4 \\ 4 \end{bmatrix}$ with label "FALSE" (not octopus)

Output is $4+4-1 \geq 0$ = TRUE. Error is -1.

$x' = 1 + 0.5(-1)4 = 1 - 2 = -1$ $\quad y' = 1 + 0.5(-1)4 = -1$

$b' = -1 + 0.5(-1) = -1.5$ New equation $-x - y - 1.5 \geq 0$

$-x - y - 1.5 \geq 0$ $\quad y \leq -x - 1.5$



Ok, but we only have one datapoint and no octopus examples...

New datapoint: $\begin{bmatrix} 8 \\ 3 \end{bmatrix}$ and it's true (OCTOPUS)

We enter $-8 - 3 - 1.5 = -12.5$ and get FALSE.

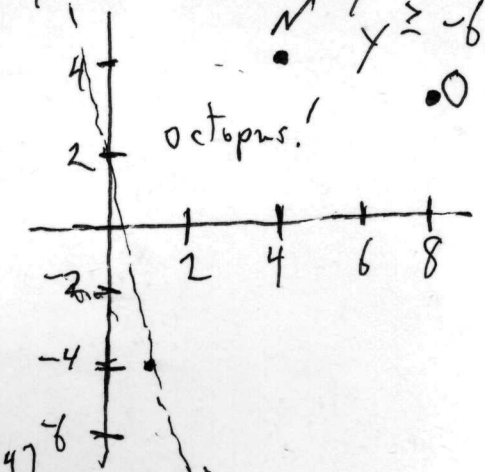
Error is 1.

$x' = -1 + 0.5(1)(8) = 3$ $\quad y' = -1 + 0.5(1)(3) = 0.5$

$b' = -1.5 + 0.5(1) = -1$

New surface $3x + 0.5y - 1 \geq 0 \Rightarrow 0.5y \geq -3x + 1$

$y \geq -6x + 2$



Which may not seem like progress, but our line is getting steeper because of this example, and the y-intercept increased.

We can actually continue to get mileage out of old examples as long as we haven't converged. Datapoint $\begin{bmatrix} 4 \\ 4 \end{bmatrix}$ again:

output $3(4) + 0.5(4) - 1 \geq 0$

$13 \geq 0$ TRUE but datapoint labeled FALSE.

Error -1:

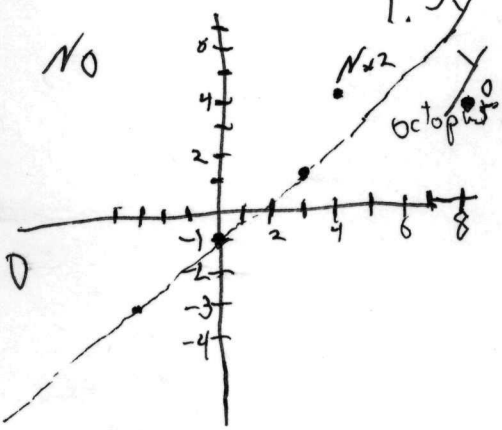
$x' = 3 + 0.5(-1)4 = 1$ $\quad y' = 0.5 + 0.5(-1)4 = -1.5$

$b' = -1 + 0.5(-1) = -1.5$ $\quad x - 1.5y - 1.5 \geq 0$

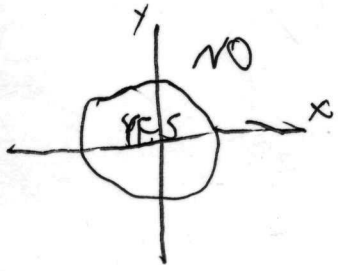
$1.5y \leq x - 1.5$

$y \leq \frac{2}{3}x - 1$ $\quad \frac{1}{\frac{3}{2}} = \frac{2}{3}$

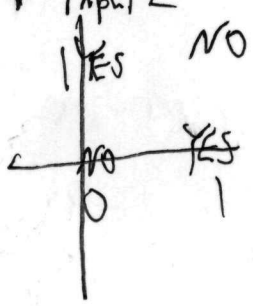
Now we actually have a classifier that will work for our examples so far. It won't move once err=0 for all examples.



The perceptron learning rule definitely has concepts it can't learn, because the separating boundary isn't flat.

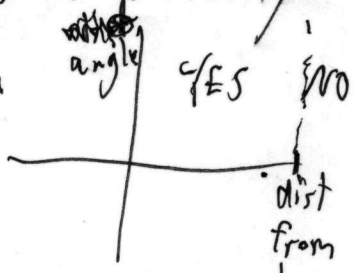


"within 1 mile of home"



XOR

But if the features can be transformed into a space where the boundary is linear, the perceptron is a perfectly good classifier,



[Optional neuroscience as time permits]

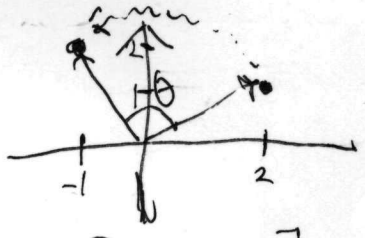
Week 2

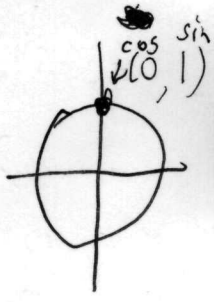
(Hefferon II.I – II.III)

More on transformations

Identity first: rot by 0, scale by 1, don't permute $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

We've seen that ~~position~~ vectors can be seen as points in a space. We can transform those points in a variety of ways with matrix multiplication.





2D rotation: $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

For example, if $\theta = 90^\circ$ or $\frac{\pi}{2}$,

$$R = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

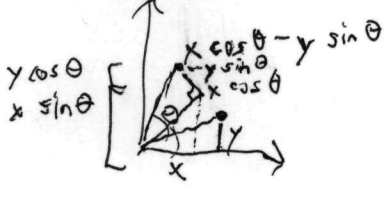
$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

(see above)

This comes from the equations $x' = x\cos\theta - y\sin\theta$

$y' = x\sin\theta + y\cos\theta$

which in turn come from the geometry of the situation



Rotations in 3 dimensions

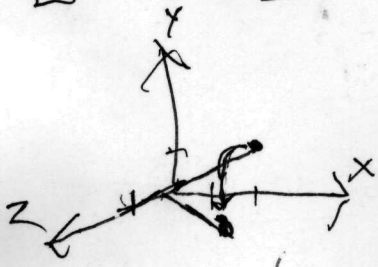
We can rotate in any 3 dimensions by performing a 2D rotation while keeping one coordinate fixed.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

90° for each:

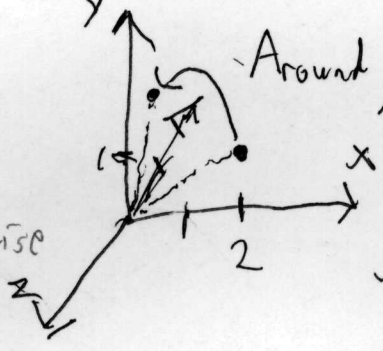
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 0 \end{bmatrix}$$

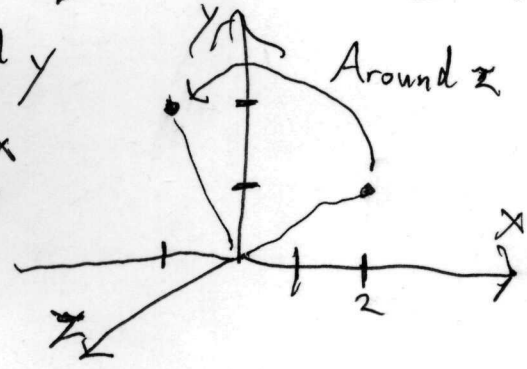


Around x, x toward +z

(convention: counterclockwise if axis is pointing to you)



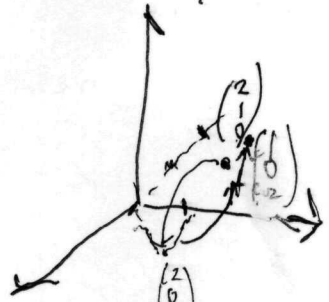
Around y



Around z

To achieve a more complex rotations, we can combine these simple rotations. To rotate 90° around x and then 90° around y.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}$$



We could compose these transformations into a single rotation matrix by multiplying them out ahead of time:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix}$$

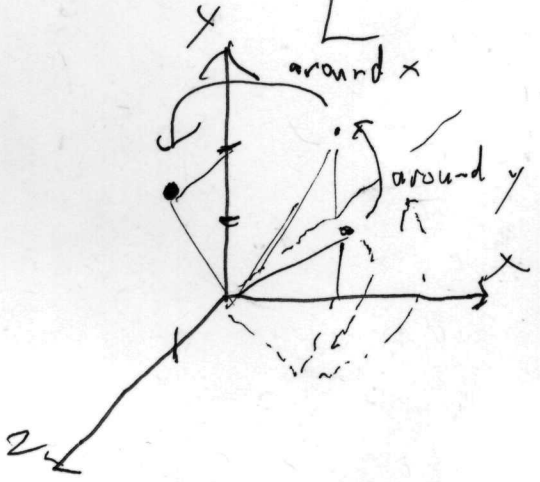
$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}$$

In general matrix multiplication reflects the way linear transformations can be combined/composed to create transformations that combine the effects of the individual transformations. Matrix and vector multiplication is associative, so $A(B\vec{v}) = (AB)\vec{v}$.

↳ Rotations also illustrate that matrix multiplication is not commutative: $AB \neq BA$. Order matters for rotations on different axes.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \neq \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}$$



Scaling is a bit simpler.

First, notice a matrix with 1's on the diagonal and 0's elsewhere — the identity — leaves vectors alone:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix}$$

We can get a matrix that scales by multiplying the identity by our scaling constant,

$$\begin{bmatrix} 5 & 0 & 0 & \cdots & 0 \\ 0 & 5 & 0 & \cdots & 0 \\ 0 & 0 & 5 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 5 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix} = \begin{bmatrix} 5v_1 \\ 5v_2 \\ \vdots \\ 5v_N \end{bmatrix}$$

Permutations are like the identity, but we swap around which value goes where. Each row and each column has one 1 and the rest zeros.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 2 \\ 5 \\ 4 \end{bmatrix}$$

These are a few of the transformations that can be expressed as matrices - and composed through multiplications.

Thinking abstractly about vectors again

Like modular arithmetic, math on vectors can be studied on its own terms for what is legal algebraically. Some properties of these systems called vector spaces: (set of vectors, $+$, $*$)

- Closed under addition
- Addition is commutative; $\vec{v} + \vec{w} = \vec{w} + \vec{v}$
- Addition is associative; $\vec{u} + (\vec{v} + \vec{w}) = (\vec{u} + \vec{v}) + \vec{w}$
- Addition has an identity $\vec{0}$ and inverses $\vec{v} + \vec{w} = 0$
- Closed under mult by a scalar
- Addition of scalars distributes; $(r+s)\vec{v} = r\vec{v} + s\vec{v}$
- Addition of vectors distributes; $(\vec{v} + \vec{w})r = r\vec{v} + r\vec{w}$

There isn't just one vector space that these facts apply to. These facts are true of $\mathbb{R}^3$ or 3D space. They're also true of $\mathbb{R}^2$ 2D space, or even the trivial space of $\{\vec{0}\}$. This also shows vector spaces can contain smaller spaces within them.

A subspace is a vector space where its vectors are a subset of another space. Not every subset is a subspace, because not all subsets are closed under addition. $\{\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix}\}$ is not a vector space because I can add them to get $\begin{pmatrix}1\\2\end{pmatrix}$, not in the space.

Because of this requirement to be closed, a nonempty subset will be a subspace if and only if all linear combinations of its elements are also in the subset, ~~(we could add items to themselves~~ (we need the multiples because we're closed under scalar mult too.)

If what we have as a subset is not yet a subspace, the span of the subset will be. It's the set of all linear combinations of the subset.

p.94 So the span of $\{\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix}\}$ is in fact $\mathbb{R}^2$, which is a vector space.

Linear independence

This is a notion that helps us understand when systems of equations will have unique solutions.

Vectors are linearly independent if no vector is a linear combination of the other vectors. (If the opposite is true, it's "linearly dependent." Removing a vector from a set of linearly independent vectors will shrink the span, at the very least, that vector itself can't be reconstructed. And if it could be reconstructed, then it wasn't linearly independent.

Similarly, if a vector not in the span is added, the span will grow.

So the linearly independent vectors are a minimal set of vectors that define a vector space. Add a vector that's not in the span, and the span increases. Add one that is in the span, and the set's not minimal.

Subsets of linearly independent sets are also linearly independent; you still can't construct one from the others.

Basis & Dimension

One use of linear independence is to understand how many dimensions we're really working with, and to create more simple vectors to characterize our position in a space.

A basis for a vector space is a sequence of vectors that is linearly independent and spans the space. (It's a sequence instead of a set just so we can have a coordinate system based on the basis, though this is a minor point.) The "standard basis" for $\mathbb{R}^n$ is $\langle \begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix}, \begin{pmatrix}0\\1\\\vdots\\0\end{pmatrix}, \ldots, \begin{pmatrix}0\\0\\\vdots\\1\end{pmatrix} \rangle$ sometimes denoted $\langle \vec{e}_1, \ldots, \vec{e}_n \rangle$

n vectors

A basis can be infinite for a space with infinite dimensions.

⇒ A subset is a basis iff each vector in the space can be described as a linear combo of its vectors in exactly one way. Pf sketch: If not, then $c_1\vec{\beta}_1 + c_2\vec{\beta}_2 + \ldots + c_n\vec{\beta}_n = d_1\vec{\beta}_1 + d_2\vec{\beta}_2 + \ldots + d_n\vec{\beta}_n$, so $(c_1-d_1)\vec{\beta}_1 + \ldots + (c_n-d_n)\vec{\beta}_n = \vec{0}$. We can't cancel one $\vec{\beta}_i$ term with a linear combo of the others, or else we could also express $\vec{\beta}_i$ as a linear combo by multiplying that by a constant — illegal. So $c_i = d_i$

for all i, and there must be only one way to express the vector.

So there's only one way to represent a vector as a weighted sum of these vectors, and that means we can use those weights as an unambiguous coordinate system. For example with basis $\left\langle \begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix} \right\rangle$ we can represent the vector (or point) $\begin{pmatrix}3\\5\end{pmatrix}$ with the coordinate vector $\begin{pmatrix}3\\1\end{pmatrix}$ since $3\begin{pmatrix}1\\1\end{pmatrix} + 1\begin{pmatrix}0\\2\end{pmatrix} = \begin{pmatrix}3\\5\end{pmatrix}$.

We can in general solve for what these coordinates should be with algebra - for example, for $\begin{pmatrix}4\\5\end{pmatrix}$ we'd have

$$c_1\begin{pmatrix}1\\1\end{pmatrix} + c_2\begin{pmatrix}0\\2\end{pmatrix} = \begin{pmatrix}4\\5\end{pmatrix} \quad \begin{matrix} c_1 = 4 \\ c_1 + 2c_2 = 5 \end{matrix} \quad 2c_2 = 1 \quad c_2 = \frac{1}{2}$$

We could prove that any basis of the same vector space would have the same # of basis vectors. We can call this the dimension of the space without worrying about which basis we're using. This is also the maximum number of linearly independent vectors we could have.

Rank

When we analyze matrices we'll want to know things like, would equations corresponding to this matrix have a unique solution? The rank is a major tool in this analysis.

We can break a matrix down into vectors in two ways. One is to think of each row as a vector.

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} = \{(1\ 2), (3, 4)\}$$

And the other is to break down by column: $\begin{pmatrix}1\\3\end{pmatrix}, \begin{pmatrix}2\\4\end{pmatrix}$