Dot-product

Dot-product of two D-vectors is sum of product of corresponding entries:

$$\mathbf{u} \cdot \mathbf{v} = \sum_{k \in D} \mathbf{u}[k]\ \mathbf{v}[k]$$

Example: For traditional vectors $\mathbf{u} = [u_1, \ldots, u_n]$ and $\mathbf{v} = [v_1, \ldots, v_n]$,

$$\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n$$

Output is a scalar, not a vector
Dot-product sometimes called *scalar product*.

Dot-product

Example: Dot-product of $[1, 1, 1, 1, 1]$ and $[10, 20, 0, 40, -100]$:

$$
\begin{array}{rrrrrrrrrrr}
 & 1 & & 1 & & 1 & & 1 & & 1 & & \\
\bullet & 10 & & 20 & & 0 & & 40 & & \text{-100} & & \\
\hline
 & 10 & + & 20 & + & 0 & + & 40 & + & (\text{-100}) & = & \text{-30}
\end{array}
$$

Quiz: Dot-product

Quiz: Write a procedure `list_dot(u, v)` with the following spec:

- *input:* equal-length lists `u` and `v` of field elements
- *output:* the dot-product of `u` and `v` interpreted as vectors

Hint: Use the `sum(·)` procedure together with a list comprehension.

Quiz: Dot-product

Quiz: Write a procedure `list_dot(u, v)` with the following spec:

- *input:* equal-length lists `u` and `v` of field elements
- *output:* the dot-product of `u` and `v` interpreted as vectors

Hint: Use the `sum(·)` procedure together with a list comprehension.

Answer:

```
def list_dot(u, v): return sum([u[i]*v[i] for i in range(len(u))])
```

or

```
def list_dot(u, v): return sum([a*b for (a,b) in zip(u,v)])
```

Dot-product: Total cost or benefit

Suppose D consists of four main ingredients of beer:

$$D = \{\text{barley}, \text{hops}, \text{yeast}, \text{water}\}$$

A *cost* vector maps each food to a price per unit amount:

$$cost = \{\text{barley} : \$0.5, \text{hops} : \$0.5, \text{yeast} : \$.28/g, \text{water} : \$0.05\}$$

A *quantity* vector maps each food to an amount (e.g. measured in pounds).
$quantity = \{$barley:0.5 lbs, hops:0.2 oz, yeast:2.5 g, water:10 gallons$\}$

The total cost is the dot-product of *cost* with *quantity*:

$$cost \cdot quantity = \$0.5 \cdot 0.5 + \$0.5 \cdot 0.2 + \$0.28 \cdot 2.5 + \$0.05 \cdot 10 = \$1.55$$

A *value* vector maps each food to its caloric content per pound:

$$value = \{\text{barley} : 544, \text{hops} : 101, \text{yeast} : 83, \text{water} : 0\}$$

The total calories represented by a pint of beer is the dot-product of *value* with *quantity*:

$$value \cdot quantity = 544 \cdot 0.5 + 101 \cdot 0.5 + 83 \cdot 2.5 + 0 \cdot 10 = 530$$

Dot-product: Linear equations

Example: A sensor node consist of hardware components, e.g.

- CPU
- radio
- temperature sensor
- memory

Battery-driven and remotely located so we care about energy usage.

Suppose we know the current draw for each hardware component.
Represent it as a D-vector with $D = \{radio, sensor, memory, CPU\}$

$$\mathbf{rate} = \{radio : 500\text{mA}, sensor : 250\text{mA}, memory : 100\text{mA}, CPU : 300\text{mA}\}$$

Have a test period during which we know how long each component was working.
Represent as another D vector:

$$\mathbf{duration} = \{radio : 0.2\text{s}, sensor : 0.5\text{s}, memory : 1.0\text{s}, CPU : 1.0\text{s}\}$$

Total milliampere-seconds:

$$\mathbf{duration} \cdot \mathbf{rate}$$

Dot-product: Linear equations

Turns out: We can only measure *average current drawn by the sensor node* over a period

Goal: calculate how much current is drawn by each hardware component.

Challenge: Cannot simply turn on memory without turning on CPU.
Idea:

- Run several tests on sensor node in which we measure total current flow
- In each test period, we know the duration each hardware component is turned on. For example,

$$\mathbf{duration}_1 = \{\mathit{radio} : 0.2\text{s}, \mathit{sensor} : 0.5\text{s}, \mathit{memory} : 1.0\text{s}, \mathit{CPU} : 1.0\text{s}\}$$
$$\mathbf{duration}_2 = \{\mathit{radio} : 0\text{s}, \mathit{sensor} : 0.1\text{s}, \mathit{memory} : 0.2\text{s}, \mathit{CPU} : 0.5\text{s}\}$$
$$\mathbf{duration}_3 = \{\mathit{radio} : .4\text{s}, \mathit{sensor} : 0\text{s}, \mathit{memory} : 0.2\text{s}, \mathit{CPU} : 1.0\text{s}\}$$

- In each test period, we know the total current flow: $\beta_1 = 1, \beta_2 = 0.75, \beta_3 = .6$
- Use data to calculate current for each hardware component.

Dot-product: Linear equations

A *linear equation* is an equation of the form

$$\mathbf{a} \cdot \mathbf{x} = \beta$$

where **a** is a vector, β is a scalar, and **x** is a vector of variables.

In sensor-node problem, we have linear equations of the form

$$\mathbf{duration}_i \cdot \mathbf{rate} = \beta_i$$

where **rate** is a vector of variables.

Questions:

- Can we find numbers for the entries of **rate** such that the equations hold?
- If we do, does this guarantee that we have correctly calculated the current draw for each component?

Dot-product: Linear equations

More general questions:

- Is there an algorithm for solving a *system of linear equations*?

$$\begin{aligned} \mathbf{a}_1 \cdot \mathbf{x} &= \beta_1 \\ \mathbf{a}_2 \cdot \mathbf{x} &= \beta_2 \\ &\vdots \\ \mathbf{a}_m \cdot \mathbf{x} &= \beta_m \end{aligned}$$

- How can we know whether there is only one solution?
- What if our data are slightly inaccurate?

These questions motivate much of what is coming in future weeks.

Dot-product: Measuring similarity: Comparing voting records

Can use dot-product to measure similarity between vectors.
Upcoming lab:

- Represent each senator's voting record as a vector:

$$[+1, +1, 0, -1]$$

$$+1 = \textit{In favor},\ 0 = \textit{not voting},\ \text{-1} = \textit{against}$$

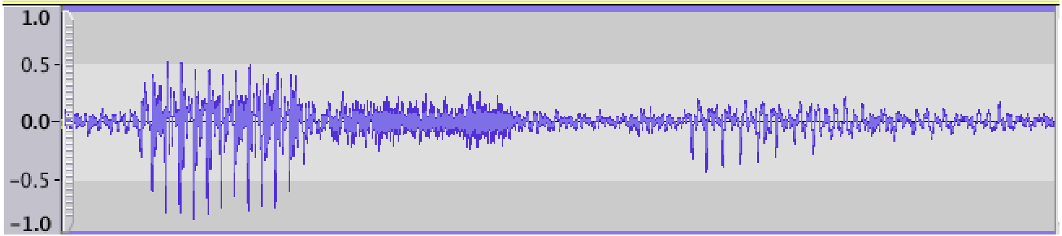
- Dot-product $[+1, +1, 0, -1] \cdot [-1, -1, -1, +1]$
  - very positive if the two senators tend to agree,
  - very negative if two voting records tend to disagree.

Dot-product: Measuring similarity: Comparing audio segments

Want to search for a short audio clip (the *needle*) in a longer audio segment (the *haystack*).

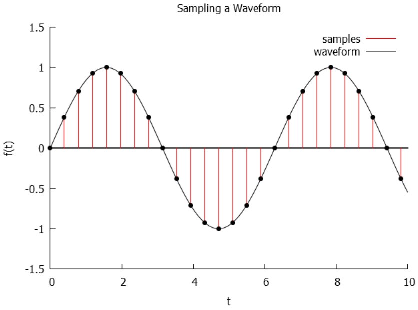
Dot-product: Measuring similarity: Comparing audio segments

Want to search for a short audio clip (the *needle*) in a longer audio segment (the *haystack*).



Dot-product: Measuring similarity: Comparing audio segments

Want to search for a short audio clip (the *needle*) in a longer audio segment (the *haystack*).



- To compare two equal-length sequences of samples, use dot-product: $\sum_{i=1}^{n} \mathbf{u}[i]\ \mathbf{v}[i]$.
- Term i in this sum is positive if $\mathbf{u}[i]$ and $\mathbf{v}[i]$ have the same sign, and negative if they have opposite signs.
- The greater the agreement, the greater the value of the dot-product.

Dot-product: Measuring similarity: Comparing audio segments

Back to needle-in-a-haystack:

If you suspect you know where the needle is...

5	-6	9	-9	-5	-9	-5	5	-8	-5	-9	9	8	-5	-9	6	-2	-4	-9	-1	-1	-9	-3
										2	7	4	-3	0	-1	-6	4	5	-8	-9		

Dot-product: Measuring similarity: Comparing audio segments

If you don't have any idea where to find the needle, compute lots of dot-products!

5	-6	9	-9	-5	-9	-5	5	-8	-5	-9	9	8	-5	-9	6	-2	-4	-9	-1	-1	-9	-3
2	7	4	-3	0	-1	-6	4	5	-8	-9												

5	-6	9	-9	-5	-9	-5	5	-8	-5	-9	9	8	-5	-9	6	-2	-4	-9	-1	-1	-9	-3
	2	7	4	-3	0	-1	-6	4	5	-8	-9											

5	-6	9	-9	-5	-9	-5	5	-8	-5	-9	9	8	-5	-9	6	-2	-4	-9	-1	-1	-9	-3
		2	7	4	-3	0	-1	-6	4	5	-8	-9										

5	-6	9	-9	-5	-9	-5	5	-8	-5	-9	9	8	-5	-9	6	-2	-4	-9	-1	-1	-9	-3
			2	7	4	-3	0	-1	-6	4	5	-8	-9									

5	-6	9	-9	-5	-9	-5	5	-8	-5	-9	9	8	-5	-9	6	-2	-4	-9	-1	-1	-9	-3
				2	7	4	-3	0	-1	-6	4	5	-8	-9								

5	-6	9	-9	-5	-9	-5	5	-8	-5	-9	9	8	-5	-9	6	-2	-4	-9	-1	-1	-9	-3
					2	7	4	-3	0	-1	-6	4	5	-8	-9							

5	-6	9	-9	-5	-9	-5	5	-8	-5	-9	9	8	-5	-9	6	-2	-4	-9	-1	-1	-9	-3
						2	7	4	-3	0	-1	-6	4	5	-8	-9						

5	6	9	9	5	9	5	5	8	5	9	9	8	5	9	6	2	4	9	1	1	9	3

Dot-product: Measuring similarity: Comparing audio segments

Seems like a lot of dot-products—-too much computation—but there is a shortcut...
The *Fast Fourier Transform*.