Netflix intro

What data should we encode about each Netflix account holder to help us make effective recommendations?

In machine learning, clustering can be used to group similar data for prediction and recommendation. For example, each Netflix user's viewing history can be represented as a n-tuple indicating their preferences about movies in the database, where n is the number of movies in the database. People with similar tastes in movies can then be clustered to provide recommendations of movies for one another. Mathematically, clustering is based on a notion of distance between pairs of n-tuples.

Set operations

To define a set we can use the roster method, set builder notation, a recursive definition, and also we can apply a set operation to other sets.

New! Cartesian product of sets and **set-wise concatenation of sets of strings**

Definition: Let X and Y be sets. The **Cartesian product** of X and Y, denoted $X \times Y$, is the set of all ordered pairs (x, y) where $x \in X$ and $y \in Y$

$$X \times Y = \{(x, y) \mid x \in X \text{ and } y \in Y\}$$

Definition: Let X and Y be sets of strings over the same alphabet. The **set-wise concatenation** of X and Y, denoted $X \circ Y$, is the set of all results of string concatenation xy where $x \in X$ and $y \in Y$

$$X \circ Y = \{xy \mid x \in X \text{ and } y \in Y\}$$

Pro-tip: the meaning of writing one element next to another like xy depends on the data-types of x and y. When x and y are strings, the convention is that xy is the result of string concatenation. When x and y are numbers, the convention is that xy is the result of multiplication. This is (one of the many reasons) why is it very important to declare the data-type of variables before we use them.

Fill in the missing entries in the table:

Set	**Example elements in this set**:
B	A C G U
	$(\texttt{A}, \texttt{C})$ $(\texttt{U}, \texttt{U})$
$B \times \{-1, 0, 1\}$	
$\{-1, 0, 1\} \times B$	
	$(0, 0, 0)$
$\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\} \circ \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\}$	
	GGGG

Defining functions

New! Defining functions A function is defined by its (1) domain, (2) codomain, and (3) rule assigning each element in the domain exactly one element in the codomain.

The domain and codomain are nonempty sets.
The rule can be depicted as a table, formula, or English description.
The notation is

"Let the function FUNCTION-NAME: DOMAIN $\to$ CODOMAIN be given by FUNCTION-NAME(x) = $\ldots$ for every $x \in DOMAIN$".

or

"Consider the function FUNCTION-NAME: DOMAIN $\to$ CODOMAIN given by FUNCTION-NAME(x) = $\ldots$ for every $x \in DOMAIN$".

Example: The absolute value function

Domain

Codomain

Rule

Defining functions recursively

When the domain of a function is a *recursively defined set*, the rule assigning images to domain elements (outputs) can also be defined recursively.

Recall: The set of RNA strands S is defined (recursively) by:

Basis Step:	$\texttt{A} \in S, \texttt{C} \in S, \texttt{U} \in S, \texttt{G} \in S$
Recursive Step:	If $s \in S$ and $b \in B$, then $sb \in S$

where sb is string concatenation.

Definition (Of a function, recursively) A function *rnalen* that computes the length of RNA strands in S is defined by:

$$
\begin{array}{llll}
 & & rnalen : S & \to \mathbb{Z}^+ \\
\text{Basis Step:} & \text{If } b \in B \text{ then} & rnalen(b) & = 1 \\
\text{Recursive Step:} & \text{If } s \in S \text{ and } b \in B\text{, then} & rnalen(sb) & = 1 + rnalen(s)
\end{array}
$$

The domain of *rnalen* is

The codomain of *rnalen* is

Example function application:

$$rnalen(\texttt{ACU}) =$$

Extra example: A function *basecount* that computes the number of a given base b appearing in a RNA strand s is defined recursively: *fill in codomain and sample function applications*

$$
\begin{array}{llll}
 & & basecount : S \times B & \to \\
\text{Basis Step:} & \text{If } b_1 \in B, b_2 \in B & basecount(b_1, b_2) & = \begin{cases} 1 & \text{when } b_1 = b_2 \\ 0 & \text{when } b_1 \neq b_2 \end{cases} \\
\text{Recursive Step:} & \text{If } s \in S, b_1 \in B, b_2 \in B & basecount(sb_1, b_2) & = \begin{cases} 1 + basecount(s, b_2) & \text{when } b_1 = b_2 \\ basecount(s, b_2) & \text{when } b_1 \neq b_2 \end{cases}
\end{array}
$$

$basecount(\texttt{ACU}, \texttt{A}) =$

$basecount(\texttt{ACU}, \texttt{G}) =$

Extra example: The function which outputs 2^n when given a nonnegative integer n can be defined recursively, because its domain is the set of nonnegative integers.

Why represent numbers

Modeling uses data-types that are encoded in a computer.

The details of the encoding impact the efficiency of algorithms we use to understand the systems we are modeling and the impacts of these algorithms on the people using the systems.

Case study: how to encode numbers?

Base expansion definition

Definition For b an integer greater than 1 and n a positive integer, the **base b expansion of n** is

$$(a_{k-1} \cdots a_1 a_0)_b$$

where k is a positive integer, $a_0, a_1, \ldots, a_{k-1}$ are nonnegative integers less than b, $a_{k-1} \neq 0$, and

$$n = a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

Notice: *The base b expansion of a positive integer n is a string over the alphabet $\{x \in \mathbb{N} \mid x < b\}$ whose leftmost character is nonzero.*

Base b	Collection of possible coefficients in base b expansion of a positive integer
Binary ($b = 2$)	$\{0, 1\}$
Ternary ($b = 3$)	$\{0, 1, 2\}$
Octal ($b = 8$)	$\{0, 1, 2, 3, 4, 5, 6, 7\}$
Decimal ($b = 10$)	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Hexadecimal ($b = 16$)	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ letter coefficient symbols represent numerical values $(A)_{16} = (10)_{10}$ $(B)_{16} = (11)_{10}$ $(C)_{16} = (12)_{10}$ $(D)_{16} = (13)_{10}$ $(E)_{16} = (14)_{10}$ $(F)_{16} = (15)_{10}$

Base expansion examples

Common bases: Binary $b = 2$ Octal $b = 8$ Decimal $b = 10$ Hexadecimal $b = 16$

Examples:

$(1401)_2$

$(1401)_{10}$

$(1401)_{16}$

Algorithm definition

New! An algorithm is a finite sequence of precise instructions for solving a problem.

Algorithm half

Algorithm for calculating integer part of half the input

```
procedure half(n: a positive integer)
r := 0
while n > 1
   r := r + 1
   n := n − 2
return r {r holds the result of the operation}
```

n	r	$n > 1$?
6		

n	r	$n > 1$?
5		

Algorithm log

Algorithm for calculating integer part of log

```
procedure log(n: a positive integer)
r := 0
while n > 1
   r := r + 1
   n := half(n)
return r {r holds the result of the log operation}
```

n	r	$n > 1$?
8		

n	r	$n > 1$?
6		

$2^0 = 1 \quad 2^1 = 2 \quad 2^2 = 4 \quad 2^3 = 8 \quad 2^4 = 16 \quad 2^5 = 32 \quad 2^6 = 64 \quad 2^7 = 128 \quad 2^8 = 256 \quad 2^9 = 512 \quad 2^{10} = 1024$

Division algorithm

Integer division and remainders (aka The Division Algorithm) Let n be an integer and d a positive integer. There are unique integers q and r, with $0 \leq r < d$, such that $n = dq + r$. In this case, d is called the divisor, n is called the dividend, q is called the quotient, and r is called the remainder. We write $q = n$ **div** d and $r = n$ **mod** d.

Extra example: How do **div** and **mod** compare to / and % in Java and python?

Base expansion algorithms

Two algorithms for constructing base b expansion from decimal representation

Most significant first: Start with left-most coefficient of expansion

Calculating integer part of $\log_b$

```
procedure logb(n, b: positive integers with b > 1)
r := 0
while n > 1
   r := r + 1
   n := n div b
return r {r holds the result of the log_b operation}
```

Calculating base b expansion, from left

```
procedure baseb1(n, b: positive integers with b > 1)
v := n
k := logb(n, b) + 1
for i := 1 to k
   a_{k-i} := 0
   while v ≥ b^{k-i}
      a_{k-i} := a_{k-i} + 1
      v := v - b^{k-i}
return (a_{k-1}, ..., a_0) {(a_{k-1} ... a_0)_b is the base b expansion of n}
```

Least significant first: Start with right-most coefficient of expansion

Idea: (when $k > 1$)

$$n = a_{k-1}b^{k-1} + \cdots + a_1 b + a_0$$
$$= b(a_{k-1}b^{k-2} + \cdots + a_1) + a_0$$

so $a_0 = n$ **mod** b and $a_{k-1}b^{k-2} + \cdots + a_1 = n$ **div** b.

Calculating base b expansion, from right

```
procedure baseb2(n, b: positive integers with b > 1)
q := n
k := 0
while q ≠ 0
   a_k := q mod b
   q := q div b
   k := k + 1
return (a_{k-1}, ..., a_0) {(a_{k-1} ... a_0)_b is the base b expansion of n}
```