Learn Haskell and R by doing Data Mining

Mihai Maruseac

June 9, 2014

The Languages

Haskell	functional programming, multipurpose
R	statistical computing and graphics

The Interpreter

Haskell `ghci`

- can compile with `ghc` (`--make`)
- can have build system: `cabal`
- `:l`, `:re`, `:t`, `:i`

R `R`

- compilable with special packages (`jit`, `compiler`)
- `source(file)`

The Layout rules

Haskell	indentation or curly braces and ;
R	indentation or curly braces and ;

The Types

Haskell static typing

Bool	`True` or `False`
Int	Integer (also `Integer`, `Float`, ..)
Char	Character
String	String (list of characters)
a	type variable
[a]	list of *a*s (types)
(a, b)	pair of an *a* and a *b* (type variables)

R dynamic typing

- `as.array`, `is.array`, `array`

The Vectors (Lists, Arrays)

Haskell Lists only

- Vectors, etc. defined in packages
- Matrices: lists of lists (inefficient)
- constructors: `[]` and `:`
- `3 : [2, 4, 1]` is the same as `[3, 2, 4, 1]`
- `[1 .. 10]`, `[1, 3 .. 10]`, `[1..]`
- `head`, `tail`, `list !! index`
- `drop`, `take`
- **Lazy evaluation**

R construct everything from vectors (`array`, `vector`, `list`, `dataframe`, `matrix`)

- `c`, empty constructor, `1:10`
- only `1:10`
- `head`, `tail`, `vector[index]`

Haskell :: More about Haskell Types

- write your own type:
  - type synonyms: `type String = [Char]`
  - new data types: `data Type = Constructor ...`
  - type variables in constructing types (think generics)
    `data Maybe a = Nothing | Just a`
    `data Either a b = Left a | Right b`
- algebraic types:
  - sum types: `data MyType = C1 Bool | C2 Char`
  - product types: `(Bool, Char)` or `data MyType = C Bool Char`
  - exponential types: functions

The Functions

Haskell

- first class citizens (expressions)
- static typing: $a \rightarrow b$
- types as:
  - documentation
  - helpers
  - proofs
- `function arg1 arg2`
- `.` for function composition
- `f $ x = f x`
- get rid of All parens!

R

- first class citizens too
- dynamic typing, use `is.*` and `as.*`
- `function(arg1, arg3=value)`

Overloading

Haskell	based on typeclasses (next slide)
R	redefine the function

Haskell :: Typeclasses

- think Java interfaces
- methods available to all members of the class

```
class Show a where
  show ::  a -> String
```

Haskell :: Typeclasses

- think Java interfaces
- methods available to all members of the class

```
class Show a where
  show ::  a -> String
```

```
class Eq a where
  (==) ::  a -> a -> Bool
  (/=) ::  a -> a -> Bool
  x == y = not $ x /= y
  x /= y = not $ x == y
```

- but with default methods

Haskell :: Some Important Typeclasses

`Show`	`show` (think `toString`)
`Read`	`read`
`Eq`	`==` and `/=`
`Ord`	`compare`, $<$, ...
`Enum`	
`Bounded`	
`Num`	
`Integral`	

Haskell :: Registering Type to Typeclass

```
data MyType = X ..  | Y ..  deriving (Eq, Show)
```

Haskell :: Registering Type to Typeclass

```
data MyType = X ..  | Y ..  deriving (Eq, Show)
```

```
data MyType = X ..  | Y ..
instance Show MyType where
   show (X ..)  = ..
   show (Y ..)  = ..
```

The Ifs

Haskell `if e1 then e2 else e3`

- both branches needed
- `e1 :: Bool`
- `e2` and `e3` have same type

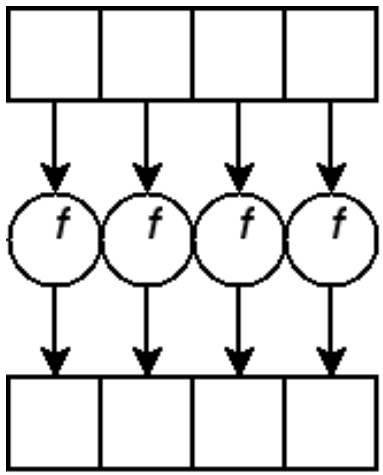
R same as in C, Java, etc.

The Loops

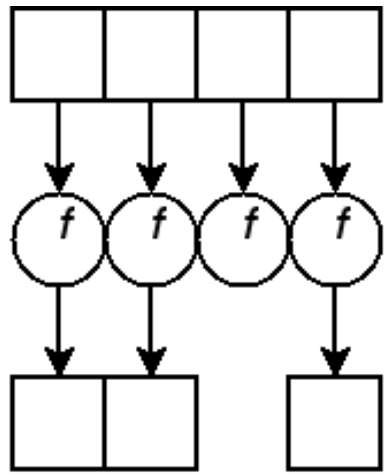
Haskell recursion or higher order functions

R same as in C, Java, etc. or higher order functions (better avoided)

Haskell :: Higher Order Functions

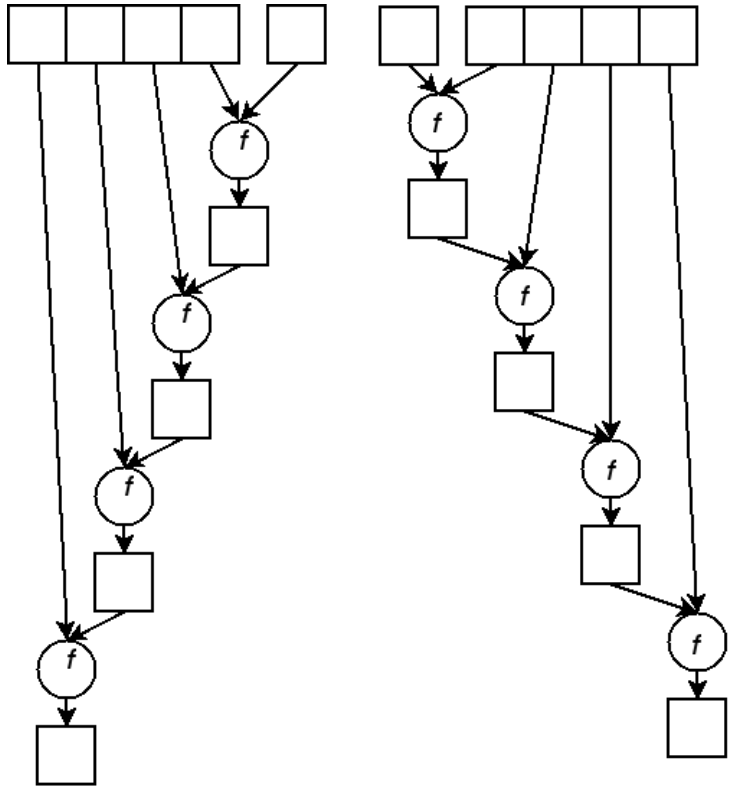


`map`



`filter`

Haskell :: Higher Order Functions



`foldr` `foldl`

R :: Array ops

- `names`, `colnames`, `rownames`
- ranges: `array[min:max]`
- range removals: `array[-min:-max]`
- item comparisons: `vector` $\leq$ `bound`
- boolean selection: `vector[condition]`

Other interesting functions

Haskell `replicate`, `iterate`, `repeat`, `dropWhile`, ...

R `replicate`

Random number generation

Haskell `System.Random`

R `sample`

Haskell :: Random

In `System.Random`

- `random :: (RandomGen g, Random a) => g -> (a, g)`
- `mkStdGen :: Int -> StdGen`
- `randomRs :: (RandomGen g, Random a) => (a, a) -> g -> [a]`

Haskell :: random number generation

```
import System.Random

data Bit = Zero | One deriving (Eq, Show, Enum, Bounded)

instance Random Bit where
  random g = (b, g') where
    (v, g') = randomR (0, 1) g
    b = if v == (0 :: Int) then Zero else One
  randomR (a, b) g = (x, g') where
    (y, g') = randomR (fromEnum a, fromEnum b) g
    x = toEnum y

getRandomBits :: StdGen -> Int -> [Bit]
getRandomBits g n = take n $ randomRs (minBound, maxBound)
```

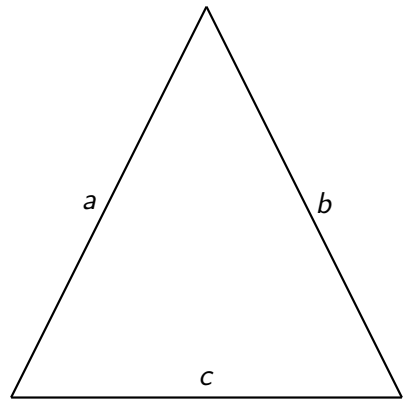
Distance metric

$$
\begin{aligned}
d(x,y) &\geq 0 \\
d(x,y) = 0 &\Leftrightarrow x = y \\
d(x,y) &= d(y,x) \\
d(x,y) &\leq d(x,z) + d(z,y)
\end{aligned}
$$

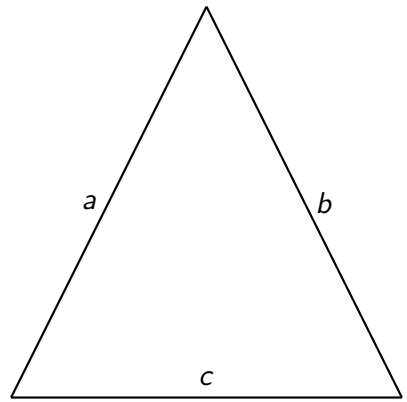
Distance ultrametric

$$
\begin{aligned}
d(x,y) &\geq 0 \\
d(x,y) = 0 &\Leftrightarrow x = y \\
d(x,y) &= d(y,x) \\
d(x,y) &\leq \max\{d(x,z), d(z,y)\}
\end{aligned}
$$

All triangles are issosceles



All triangles are issosceles



$$
\begin{aligned}
a &\geq c \\
b &\leq \max\{a, c\} \\
b &\leq a \\
a &\leq \max\{b, c\}
\end{aligned}
$$

Other relations

- all points in a sphere are centers of the sphere

Other relations

- all points in a sphere are centers of the sphere
- two spheres with a common point are the same sphere

Ultrametric clustering

1. select a random item x from the set of items
2. for all other items y, compute $d(x, y)$
3. all items y closer to x than a threshold a form one cluster
4. repeat all steps until there are no more items

Tasks

1. Write a function to generate an ADN sequence of sz nucleotides (A, C, G, T)
2. Write a function to generate n sequences of size sz
3. Write a function to compute the following distance between two sequences: if the nucleotide at position i is different and all nucleotides before are equal then the distance is 2^{-i} (thus, a distance between 0 and 1)
4. Do the clustering