Contents

1 Module Util : Various utility types and functions of my own devising

```
module Hashtbl :
   ExtHashtbl.Hashtbl
module Array :
   ExtArray.Array
module String :
   ExtString.String
module List :
   ExtList.List
module Opt :
   Option
module Dray :
   DynArray
val (+=) : int Pervasives.ref -> int -> unit
val (-=) : int Pervasives.ref -> int -> unit
val (*=) : int Pervasives.ref -> int -> unit
val (/=) : int Pervasives.ref -> int -> unit
val (+.=) : float Pervasives.ref -> float -> unit
val (-.=) : float Pervasives.ref -> float -> unit
val (*.=) : float Pervasives.ref -> float -> unit
val (/.=) : float Pervasives.ref -> float -> unit
val (|>) : 'a -> ('a -> 'b) -> 'b
val (>>) : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b
val ($) : ('a -> 'b) -> 'a -> 'b
val swap : ('a -> 'b -> 'c) -> 'b -> 'a -> 'c
val flip : ('a -> 'b -> 'c) -> 'b -> 'a -> 'c
type label = int
type features = (int, float) Hashtbl.t
type meta = {
  global : string ;
  local : string ;
}
```

```
type instance = {
  label : int ;
  features : features ;
  info : meta ;
}
type data = instance list
val empty_instance : instance
val list_of_array : 'a array -> 'a list
val array_of_list : 'a list -> 'a array
val bool_of_int : int -> bool
val int_of_bool : bool -> int
val boi : int -> bool
val iob : bool -> int
val sob : bool -> string
val bos : string -> bool
val soi : int -> string
val sof : float -> string
val ios : string -> int
val fos : string -> float
val iof : float -> int
val foi : int -> float
val loa : 'a array -> 'a list
val aol : 'a list -> 'a array
val soc : char -> string
val aoe : 'a Enum.t -> 'a array
val eoa : 'a array -> 'a Enum.t
val loe : 'a Enum.t -> 'a list
val eol : 'a list -> 'a Enum.t
val ioc : char -> int
val coi : int -> char
val rev_compare : 'a -> 'a -> int
module A :
  sig

    include Array
    val iter2 : ('a -> 'b -> 'c) -> 'a array -> 'b array -> unit
    val iter2i : (int -> 'a -> 'b -> 'c) -> 'a array -> 'b array -> unit
    val switer : 'a array -> ('a -> unit) -> unit
    val switeri : 'a array -> (int -> 'a -> unit) -> unit
```

```
val switer2 : 'a array -> 'b array -> ('a -> 'b -> 'c) -> unit
val switer2i : 'a array -> 'b array -> (int -> 'a -> 'b -> 'c) -> unit
val map2 : ('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array
val map_matrix : ('a -> 'b) -> 'a array array -> 'b array array
val map_2D : ('a -> 'b) -> 'a array array -> 'b array array
val mapij_2D : (int -> int -> 'a -> 'b) -> 'a array array -> 'b array array
val iter_2D : ('a -> 'b) -> 'a array array -> unit
val iterij_2D : (int -> int -> 'a -> 'b) -> 'a array array -> unit
val iter_3D : ('a -> 'b) -> 'a array array array -> unit
val iterijk_3D :
  (int -> int -> int -> 'a -> 'b) -> 'a array array array -> unit
val smap : 'a array -> ('a -> 'b) -> 'b array
val smapi : 'a array -> ('a -> 'b) -> 'b array
val smap2 : 'a array -> 'b array -> ('a -> 'b -> 'c) -> 'c array
val shuffled_in : ?seed:int option -> 'a array -> 'a array
```

in-place shuffle

```
val shuffled : ?seed:'a option -> 'b array -> 'b array
```

non-destructive shuffle

```
val shuffle : 'a array -> unit
```

imperative (destructive) shuffle

```
val put : (int -> 'a) -> 'a array -> unit
```

`put f a` replaces the `i`th element of array `a` with `f i`

```
val update : ('a -> 'a) -> 'a array -> unit
```

`update f a` replaces each element `e` in array `a` with `f e`

```
val unique : 'a array -> 'a array
```

`unique a` returns a new array `a'` with all the unique elements of `a` appearing exactly once, in the intial order of appearance in `a`.

```
val to_ht : 'a array -> ('a, int) Util.Hashtbl.t
```

`to_ht a` returns a hashtable `h` where `Ht.find h elt = i`, for some `i` s.t. `a.(i) = elt`

```
val except : int -> 'a array -> 'a array
```

`except i a` returns a copy of array `a` excluding element index `i`

```
val find_item : 'a -> 'a array -> int
val find_alli : ('a -> bool) -> 'a array -> int array
```

`find_alli p a` returns the index array of elements from array `a` that satisfy predicate `p`

```
val existsi : (int -> 'a -> bool) -> 'a array -> bool
val slice : ?first:int -> ?last:int -> 'a array -> 'a array
val split : 'a array -> int -> 'a array * 'a array
val range : int -> int -> int array
```

`range fst lst` returns an array of ints from `fst` inclusive to `lst` exclusive

```
val findi_next : int -> ('a -> bool) -> 'a array -> int
val find_next : int -> ('a -> bool) -> 'a array -> 'a
val rand_iter : ?seed:int option -> ('a -> 'b) -> 'a array -> unit
val rand_iteri : ?seed:int option -> (int -> 'a -> 'b) -> 'a array -> unit
val of_str :
  btw:string -> l:string -> r:string -> (string -> 'a) -> string -> 'a array
val of_str1 : (string -> 'a) -> string -> 'a array
val of_str2 : (string -> 'a) -> string -> 'a array
val to_str :
  btw:string -> l:string -> r:string -> ('a -> string) -> 'a array -> string
val to_str1 : ('a -> string) -> 'a array -> string
val to_str2 : ('a -> string) -> 'a array -> string
val all_same : 'a array -> bool
val normalize : float array -> unit
val count : 'a array -> 'a -> int
val incr : int array -> int -> unit
val decr : int array -> int -> unit
val incr' : float array -> int -> unit
val decr' : float array -> int -> unit
val fill_all : 'a array -> 'a -> unit
val init_matrix : int -> int -> (int -> int -> 'a) -> 'a array array
val init_2D : int -> int -> (int -> int -> 'a) -> 'a array array
val init_symm_matrix : int -> (int -> int -> 'a) -> 'a array array
val make_2D : int -> int -> 'a -> 'a array array
val make_3D : int -> int -> int -> 'a -> 'a array array array
val init_3D :
  int -> int -> int -> (int -> int -> int -> 'a) -> 'a array array array
val fill_all_matrix : 'a array array -> 'a -> unit
val fill_all_2D : 'a array array -> 'a -> unit
```

```
    val fill_3D : 'a array array array -> 'a -> unit
    val copy_matrix : 'a array array -> 'a array array
    val copy_2D : 'a array array -> 'a array array
    val sum : float array -> float
    val sum_int : int array -> int
    val filteri : (int -> 'a -> bool) -> 'a array -> 'a array
    val flatten : 'a array array -> 'a array
    val concat : 'a array array -> 'a array
    val map_filter : ('a -> 'b option) -> 'a array -> 'b array
    val map_filteri : (int -> 'a -> 'b option) -> 'a array -> 'b array
    val iter_all : ('a -> unit) -> 'a array array -> unit
    val iteri_all : (int -> 'a -> unit) -> 'a array array -> unit
    val filter_all : ('a -> bool) -> 'a array array -> 'a array
    val filteri_all : (int -> 'a -> bool) -> 'a array array -> 'a array
    val map_all : ('a -> 'b) -> 'a array array -> 'b array
    val mapi_all : (int -> 'a -> 'b) -> 'a array array -> 'b array
    val map_filter_all : ('a -> 'b option) -> 'a array array -> 'b array
    val map_filteri_all : (int -> 'a -> 'b option) -> 'a array array -> 'b array
    val filter_all_ij : (int -> int -> 'a -> bool) -> 'a array array -> 'a array
    val map_all_ij : (int -> int -> 'a -> 'b) -> 'a array array -> 'b array
    val iter_all_ij : (int -> int -> 'a -> 'b) -> 'a array array -> unit
    val map_filter_all_ij :
      (int -> int -> 'a -> 'b option) -> 'a array array -> 'b array
    val get_combos : 'a array array -> 'a array array
  end

module L :
  sig

    include List
    val iter2i : (int -> 'a -> 'b -> 'c) -> 'a list -> 'b list -> unit
    val switer : 'a list -> ('a -> unit) -> unit
    val switeri : 'a list -> (int -> 'a -> 'b) -> unit
    val switer2 : 'a list -> 'b list -> ('a -> 'b -> unit) -> unit
    val switer2i : 'a list -> 'b list -> (int -> 'a -> 'b -> 'c) -> unit
    val smap : 'a list -> ('a -> 'b) -> 'b list
    val smapi : 'a list -> ('a -> 'b) -> 'b list
    val smap2 : 'a list -> 'b list -> ('a -> 'b -> 'c) -> 'c list
    val all_same : 'a list -> bool
```

```
    val is_empty : 'a list -> bool
    val of_str :
      btw:string -> l:string -> r:string -> (string -> 'a) -> string -> 'a list
    val of_str1 : (string -> 'a) -> string -> 'a list
    val of_str2 : (string -> 'a) -> string -> 'a list
    val to_str :
      btw:string -> l:string -> r:string -> ('a -> string) -> 'a list -> string
    val to_str1 : ('a -> string) -> 'a list -> string
    val to_str2 : ('a -> string) -> 'a list -> string
    val count : 'a list -> 'a -> int
    val unique : 'a list -> 'a list
    val range : int -> int -> int list
    val transpose1 : 'a list list -> 'a list list
```

transposes a list of lists as if it were a matrix

```
    val transpose2 : 'a list list -> 'a list list
    val transpose : 'a list list -> 'a list list
    val map_filter : ('a -> 'b option) -> 'a list -> 'b list
  end

module Ht :
  sig
    include Hashtbl
    val add_int : ('a, int) t -> 'a -> int -> unit
    val add_float : ('a, float) t -> 'a -> float -> unit
    val incr : ('a, int) t -> 'a -> unit
    val decr : ('a, int) t -> 'a -> unit
    val incr' : ('a, float) t -> 'a -> unit
    val decr' : ('a, float) t -> 'a -> unit
    val get : ('a, 'b) t -> 'a -> 'b -> 'b
    val to_str :
      ('a -> string) ->
      ('b -> string) -> string -> string -> 'c -> string -> ('a, 'b) t -> string
    val to_str1 : ('a -> string) -> ('b -> string) -> ('a, 'b) t -> string
    val of_file : (string -> 'a) -> (string -> 'b) -> string -> ('a, 'b) t
```

`of_file key_fun val_fun fn` returns a hashtable from a file `fn`. Each line of `fn` should contain a key,value pair separated by tab

```
    val of_arrays : 'a array -> 'b array -> ('a, 'b) t
```

`of_arrays k v` returns a hashtable where `k.(i)` → `v.(i)` for each `i`.

```
val of_array : 'a array -> ('a, int) Util.Hashtbl.t
```

`of_array a` returns a hashtable `h` where `Ht.find h elt = i`, for some `i` s.t. `a.(i) = elt`

```
val to_array : ('a, 'b) t -> ('a * 'b) array
val to_sorted_array : ('a, 'b) t -> ('a * 'b) array
val to_rev_sorted_array : ('a, 'b) t -> ('a * 'b) array
val init : 'a array -> ('a -> 'b) -> ('a, 'b) t
end
module S :
  sig
    include String
    val count : string -> char -> int
    val lstrip : ?chars:string -> string -> string
    val rstrip : ?chars:string -> string -> string
    val remove : char -> string -> string
    val replace_all : str:string -> sub:string -> by:string -> string
    val nsplita : string -> string -> string array
    val concata : string -> string array -> string
    val joina : string -> string array -> string
  end
module E :
  sig
    include Enum
    val switer : 'a t -> ('a -> unit) -> unit
    val switeri : 'a t -> (int -> 'a -> unit) -> unit
    val switer2 : 'a t -> 'b t -> ('a -> 'b -> unit) -> unit
    val switer2i : 'a t -> 'b t -> (int -> 'a -> 'b -> unit) -> unit
  end
module Pair :
  sig
    val map : ('a -> 'b) -> 'a * 'a -> 'b * 'b
    val swap : 'a * 'b -> 'b * 'a
    val iter : ('a -> 'b) -> 'a * 'a -> 'b
```

```
  end
module G :
  sig
     include Global
     val notdef : 'a t -> bool
     val create : ?s:string -> 'a -> 'a t
  end
module Hs :
  sig
     include HashSet
     val of_list : 'a list -> 'a t
     val of_array : 'a array -> 'a t
     val to_array : 'a t -> 'a array
     val to_list : 'a t -> 'a list
  end
val pause : unit -> unit
val my_dump : 'a -> string
val time : (unit -> 'a) -> float
val list_of_file : string -> string list
val array_of_input : Pervasives.in_channel -> string array
val array_of_file : string -> string array
val enum_of_file : string -> string Enum.t
val output_enum : Pervasives.out_channel -> string E.t -> unit
val file_of_enum : string -> string E.t -> unit
val output_list : Pervasives.out_channel -> string list -> unit
val file_of_list : string -> string list -> unit
val output_array : Pervasives.out_channel -> string array -> unit
val file_of_array : string -> string array -> unit
val dot_prd_ff : features -> features -> float
val dot_prd_fw : features -> float array -> float
val split : ?max:int -> string -> string list
val splita : ?max:int -> string -> string array
val lines_of_str : ?comment_str:string -> string -> string array
```

`lines_of_str s` returns (XXX non-empty), non-commented lines of string `s`, and strips off comments

```
val lines_of_file : ?comment_str:string -> string -> string array
```

`lines_of_file f` returns non-empty, white-space stripped, non-commented lines of file `f`

```
val min_itemi : 'a array -> int * 'a
val min_item : 'a array -> 'a
val max_itemi : 'a array -> int * 'a
val max_item : 'a array -> 'a
val max_in_ht : ('a, 'b) Ht.t -> 'a * 'b
val min_in_ht : ('a, 'b) Ht.t -> 'a * 'b
val min_max_items : 'a array -> 'a * 'a
val factorial : Num.num -> Num.num
val fact : int -> int
val choose : int -> int -> int
val sum : float array -> float array -> float array
val round : float -> int
val parse_features : string list -> (int, float) Ht.t
val parse_data_line :
  ?global:string ->
  ?m:int -> ?single:int option -> string -> instance list
val parse_data_file :
  ?verbose:int ->
  ?multi:bool -> ?single:int option -> Pervasives.in_channel -> data * int
val get_labels : data -> label array
val get_highest_feature : data -> int
val sign : int -> int
val get_mean_dev : float list -> float * float
val id : 'a -> 'a
val deep_copy : 'a Pervasives.ref Dray.t -> 'a Pervasives.ref Dray.t
val mht_to_ht : ('a, 'b) Hashtbl.t -> ('a, 'b array) Hashtbl.t
```

Takes a multi hashtable h and returns a new Hashtable with exactly one entry for each key in h, mapped to an array of all bindings of that key in h.

```
val hs_of_list : 'a list -> 'a Hs.t
val hs_of_array : 'a array -> 'a Hs.t
val array_to_hs : 'a array -> 'a Hs.t
val output_line : Pervasives.out_channel -> string -> unit
val shuffle_list : ?seed:int option -> 'a list -> 'a list
val list_except : int -> 'a list -> 'a list
val memoize : ('a -> 'b) -> 'a -> 'b
val memoize_rec : (('a -> 'b) -> 'a -> 'b) -> 'a -> 'b
val matrix_to_str : ('a -> string) -> 'a array array -> string
```

```
val print : string -> unit
val brintf : ('a, unit, string, unit) Pervasives.format4 -> 'a
val ebrintf : ('a, unit, string, unit) Pervasives.format4 -> 'a
```

2 Module `Lattice` : The lattice module.

NB: weights on edges should represent probabilities, but they needn't sum to one. i.e. the structure of the lattice may have implicit conditioning on some outside event.

```
type node = int * int
```

`(i,j)` represents the `j`th state at time `i`

```
type 'a lattice = {
  nodes : 'a array array ;
```

`nodes.(i).(j)` represents state `j` at time `i`

```
  edges_from : (node, (node * float) array) Util.Ht.t ;
  edges_to : (node, (node * float) array) Util.Ht.t ;
  edges : (node * node, float) Util.Ht.t ;
```

edges and their weights

```
  mutable alphas : (node, float) Util.Ht.t option ;
  mutable betas : (node, float) Util.Ht.t option ;
  mutable z : float option ;
}
val mk_lattice :
  'a array array ->
  (node * node, float) Util.Ht.t -> 'a lattice
```

`mk_lattice nodes edges` makes a lattice datatype.

```
val get_viterbi_path : 'a lattice -> 'b -> node list
```

`get_viterbi_path l` returns the highest probability path through lattice `l`.

```
val get_viterbi_symbols : 'a lattice -> 'b -> 'a list
```

`get_viterbi_symbols l` returns the symbols along the highest probability path through lattice `l`.

```
val get_alphas : 'a lattice -> (node, float) Util.Ht.t
```

alpha.(i).(j) is alpha_j (i) in Mike Collins' notation i.e. alpha.(i).(j) is the sum of weights of all paths from the source to Node (i,j) `i.e. state j at time i`

```
val get_betas : 'a lattice -> (node, float) Util.Ht.t
```

beta.(i).(j) is beta_j (i) in Mike Collins' notation i.e. beta.(i).(j) is the sum of weights of all paths from Node (i,j) to the target

`val get_z : 'a lattice -> float`

Get the sum of weights of all paths through the lattice (partition function). If the lattice represents a markov model conditioned on some fact, then this will give us the probability of that fact.

`val get_node_prob : 'a lattice -> node -> float`

`get_node_prob l n` returns the probability of passing through node `n` in lattice `l`.

`val get_edge_prob : 'a lattice -> node -> node -> float`

`get_edge_prob l src trg` returns the probability of traversing the edge from node `src` to node `trg` in lattice `l`.