Implementing the Viterbi algorithm using Semigroups and folds

Introduction

The goal of this assignment is to implement the *Viterbi* algorithm using *Semigroups (or Monoids)* and *folds* in *Haskell*. A version of the *Viterbi* algorithm making use of *logs* is also required to deal with underflows. The *fold* example, given in [2], should be adapted appropriately in order to obtain an implementation of the *Viterbi* algorithm. This is an essential requirement and the project will not be accepted without this being taken into account.

Implementation Description

The V*iterbi* algorithm should be used to find the most likely sequence of hidden states, called a Viterbi path, for a sequence of observations that is observed from a *Hidden Markov Model (HMM)*.

Let S = {x_i: i = 1,2,...,K} be the set of states in the HMM and O = {y_j: j=1,2,...,T} the sequence of observations. A HMM datatype should be defined as follows, but modified to take into account that a Semigroup or Monoid is used so that the same code can handle the situation where probabilities are multiplied, or the log of probabilities are added:

```
data HMM = HMM {states :: [String],          -- State labels

            emissionLabels :: [String],      -- Emission labels

            transitionMatrix :: [[Double]],  -- Transition probability matrix

            emissionMatrix :: [[Double]],    -- Emission probability matrix

            initialProb :: [Double]          -- Initial probabilities

            } deriving (Show)
```

The Viterbi algorithm requires the use of two tables (see [1]). One for the probability of the most likely path thus far to each state, table T_1, and a table T_2 from which the states visited on each optimal path (to a given state) can be reconstructed. After T_2 is fully constructed, the Viterbi path is found by backtracking on T_2, using the state that has the highest probability in the last column of T_1 as a starting point. Note that the last column in T_1 is the only information being used for determining a starting state for backtracking and for determining the next column that should be added to T_1. We thus replace the function *roadStep* from [1] by the function *viterbiStep* having the following signature, which should again be modified to take Semigroups or Monoids into account:

viterbiStep :: HMM -> ([[Int]], [Double]) -> Int -> ([[Int]], [Double])

The types *[[Int]]* and *[Double]* are used for T_2 and T_1 respectively. Also, the *Int* in the signature indicates the next observation.

ViterbiStep (with the first argument supplied) should be used as the parameter of type function for *foldl*, and the folding should return a type *([[Int]],[a])*, where *a* is a type parameter generalizing the type *Double* in the description above.

Additional Requirement

In order to switch seamlessly between an implementation that multiplies probabilities to one that adds *logs* of probabilities, you should make use of Semigroups or Monoids. There will be a deduction of 25% if this requirement is not fulfilled.

Test cases

- The example from [1] where the observations ['normal', 'cold', 'dizzy'] are, with highest probability, generated by the states ['Healthy', 'Healthy', 'Fever'].
- Output blocks 8, 9, 10, 11 in the following Jupyter notebook.
- 10% of the final mark is reserved for additional unspecified test cases (that you should come up with on your own).

References

[1] Wikipedia: Viterbi Algorithm

[2] Functionally Solving Problems in Learn You a Haskell for Great Good!