Module Version: 0.01

NAME

Algorithm::Viterbi - Compute Viterbi path and probability

SYNOPSIS

```
use Algorithm::Viterbi;

my $start_probability = { 'Rainy'=> 0.6, 'Sunny'=> 0.4 };

my $transition_probability = {
 'Rainy' => {'Rainy'=> 0.7, 'Sunny'=> 0.3},
 'Sunny' => {'Rainy'=> 0.4, 'Sunny'=> 0.6},
};

my $emission = {
  'shop' =>  { 'Sunny' => '0.3', 'Rainy' => '0.4' },
  'walk' =>  { 'Sunny' => '0.6', 'Rainy' => '0.1' },
  'clean' => { 'Sunny' => '0.1', 'Rainy' => '0.5' }
};

my $v = Algorithm::Viterbi->new();
$v->start($start_probability);
$v->transition($transition_probability);
$v->emission($emission_probability);

my $observations = [ 'walk', 'shop', 'clean' ];

my ($prob, $v_path, $v_prob) = $v->forward_viterbi($observations);

-- or --

my $training_data = [
  [ 'walk', 'Sunny' ],
  [ 'walk', 'Sunny' ],
  [ 'walk', 'Rainy' ],
  [ 'shop', 'Rainy' ],
  [ 'clean', 'Rainy' ],
  [ 'clean', 'Rainy' ],
  ...
];

$v->train($training_data);
my ($prob, $v_path, $v_prob) = $v->forward_viterbi($observations);
```

DESCRIPTION

Algorithm::Viterbi computes the forward probability, the Viterbi path and the Viterbi probability of a sequence of observations, based on a given start, emission and transition probability. Alternatively, the start, emission and transition probability can be computed from a set of

training data.

The whole idea of this module is inspired by an article on the Viterbi algorithm in Wikipedia, the free encyclopedia. Rather than copying all text, I'm just including the link to the Wikipedia page: http://en.wikipedia.org/wiki/Viterbi_algorithm. I think the page is well-written and I see no need to repeat the theory here. Reading it may clarify the documentation below.

METHODS

new

Creates a new `Algorithm::Viterbi` object. The following attributes can be set with the constructor:

```
my $v = Algorthm::Viterbi->new(
  start_state => '$',
  unknown_emission_prob => undef,
  unknown_transition_prob => 0);
```

The values of the attributes in the example are the default values. For a detailed description and use of these attributes, see below.

train

This method computes the start, emission and transition probabilities from a set of observations and their associated states. The probabilities are simple averages of the passed observations, so if you require sophisticated smoothing on the emission, start and/or transition, then you're better off rolling your own.

The value of member start_state is a bogus state used to define the begin state of the first transition. By default, this state is set to '$'. You can change this by setting the variable in the constructor or later by accessing the member directly. See example below.

This state can also be used as a separator between the beginning and end of a sequence of observations. For example, you could assign this state (tag) to every end-of-sentence symbol when training on a pre-tagged corpus.

The set of observations is passed as a reference to an array as shown in the following example:

```
use strict;
use Algorithm::Viterbi;
use Data::Dumper;

my $observations = [
  [ 'work', 'rainy' ],
  [ 'work', 'sunny' ],
  [ 'walk', 'sunny' ],
  [ 'walk', 'rainy' ],
  [ 'shop', 'rainy' ],
  [ 'work', 'rainy' ],
];

my $v = Algorithm::Viterbi->new(start_state => '###');
$v->train($observations);

print Dumper($v);
```

will produce:

```
$VAR1 = bless( {
                 'transition' => {
                                   'sunny' => {
                                                 'sunny' => '0.5',
                                                 'rainy' => '0.25'
                                               },
                                   'rainy' => {
                                                 'sunny' => '0.5',
                                                 'rainy' => '0.5'
                                               },
                                   '###' => {
                                               'rainy' => '0.25'
                                             }
                                 },
                 'emission' => {
                                   'shop' => {
                                               'rainy' => '0.25'
                                             },
                                   'walk' => {
                                               'sunny' => '0.5',
                                               'rainy' => '0.25'
                                             },
                                   'work' => {
                                               'sunny' => '0.5',
                                               'rainy' => '0.5'
                                             }
                                 },
                 'start_state' => '###',
                 'states' => [
                               'sunny',
                               'rainy'
                             ],
                 'unknown_transition_prob' => 0,
                 'start' => {
                              'sunny' => '0.333333333333333',
                              'rainy' => '0.666666666666667'
                            }
               }, 'Algorithm::Viterbi' );
```

start

Initializes the start probabilities. The start probabilities are passed as a reference to a hash, as shown in this example:

```
my $start_probability = { 'Rainy'=> 0.6, 'Sunny'=> 0.4 };
my $v = Algorithm::Viterbi->new();
$v->start($start_probability);
```

From the start probabilities, all possible states are derived, by copying the keys of the start hash. This list of states is used by the forward_viterbi method. It is therefore important to mention all possible states in the start hash.

Returns the start probabilities.

emission

Initializes the emission probabilities. The emission is passed as a reference to a hash, as shown in this example:

```
my $emission_probability = {
        'shop' =>  { 'Sunny' => '0.3', 'Rainy' => '0.4' },
        'swim' =>  { 'Sunny' => '0.1' },
        'walk' =>  { 'Sunny' => '0.5', 'Rainy' => '0.1' },
        'clean' => { 'Sunny' => '0.1', 'Rainy' => '0.5' }
      };
my $v = Algorithm::Viterbi->new();
$v->emission($emission_probability);
```

The keys of the emission hash constitute the dictionary, which is used to determine whether an observation is a known or an unknown observation.

Returns the emission probabilities.

transition

Initializes the transition probabilities. The transition is passed as a reference to a hash, as shown in this example:

```
my $transition_probability = {
 'Rainy' => {'Rainy'=> 0.7, 'Sunny'=> 0.3},
 'Sunny' => {'Rainy'=> 0.4, 'Sunny'=> 0.6},
};
my $v = Algorithm::Viterbi->new();
$v->transition($transition_probability);
```

The transition hash can be 'sparse': it is sufficient to include only known transitions between states. See method get_transition.

Returns the transition probabilities.

forward_viterbi

This method calculates the forward probability, the Viterbi path and the Viterbi probability of a given sequence of observations. For a detailed description of the Algorithm, see the Wikipedia page http://en.wikipedia.org/wiki/Viterbi_algorithm.

The difference with the algorithm described in the web page above, is that the emission and the transition are calculated somewhat differently. See methods get_emission and get_transition.

Example:

```
use strict;
use Algorithm::Viterbi;
use Data::Dumper;


my $observations = [ 'walk', 'shop', 'clean' ];
 my $start = { 'Rainy'=> 0.6, 'Sunny'=> 0.4 };
 my $transition = {
    'Rainy' => {'Rainy'=> 0.7, 'Sunny'=> 0.3},
    'Sunny' => {'Rainy'=> 0.4, 'Sunny'=> 0.6},
    };

my $emission = {
  'shop' => {
    'Sunny' => '0.3',
```

```
      'Rainy' => '0.4',
    },

    'walk' => {
      'Sunny' => '0.6',
      'Rainy' => '0.1'
    },
    'clean' => {
      'Sunny' => '0.1',
      'Rainy' => '0.5'
     }
  };

  my $v = Algorithm::Viterbi->new();
  $v->emission($emission);
  $v->transition($transition);
  $v->start($start);

  print Dumper ($v->forward_viterbi($observations));
```

produces:

```
  $VAR1 = '0.033612';
  $VAR2 = [
            'Sunny',
            'Rainy',
            'Rainy',
            'Rainy'
          ];
  $VAR3 = '0.009408';
```

get_emission

Usage: $v->get_emission($observation, $state);

Returns the emission probability for a given observation and state. This method is primarily for internal usage and is called by the forward_viterbi method.

The dictionary consists of the keys of the emission table, e.g. a list of known observations.

If $observation is a known observation in the dictionary and $state exists as a state for the observation in the emission table, then return the probability associated with $observation and $state.

If the observation exists in the dictionary, but $state is a state not connected to the observation, then return 0.

If the observation does not exist in the dictionary and $v->{unknown_emission_prob} is defined, then return $v->{unknown_emission_prob}. Setting $v->{unknown_emission_prob} = 1 actually means that you are returning all possible states for an unknown observation.

If the observation is unknown in the dictionary and $v->{unknown_emission_prob} is not defined, then return the start probability of $state.

Example:

```
  my $emission = {
```

```
    'shop' => {
                'Sunny' => '0.3',
                'Rainy' => '0.4'
              },
    'swim' => {
                'Sunny' => '0.1'
              },
    'walk' => {
                'Sunny' => '0.5',
                'Rainy' => '0.1'
              },
    'clean' => {
                 'Sunny' => '0.1',
                 'Rainy' => '0.5'
               }
  };

  my $start = { 'Rainy'=> 0.6, 'Sunny'=> 0.4 };

  my $v = Algorithm::Viterbi->new();
  $v->emission($emission);
  $v->start($start);
  my $e;
  $e = get_emission('shop', 'Rainy'); # $e = 0.4
  $e = get_emission('swim', 'Rainy'); # $e = 0
  $e = get_emission('hack', 'Rainy'); # $e = 0.6
  $v->{unknown_emission_prob} = 1;
  $e = get_emission('hack', 'Rainy'); # $e = 1
```

get_transition

Usage: $v->get_transition($state, $next_state);

Returns the transition probability between a state and the next state. This method is primarily for internal usage and is called by the forward_viterbi method.

If the transition between $state and $next_state is defined, then return the probability associated with it.

If the transition between $state and $next_state does not exist, then return the value of $v->unknown_transition_prob, which will be 0 unless otherwise defined. Setting this attribute to a very small value allows you to still obtain a Viterbi path, although no suitable transitions were found between states of a given observation.

Example:

```
    use Algorithm::Viterbi;

    my $observations = [ 'walk', 'shop', 'read' ];

    my $start = { 'Rainy'=> 0.5, 'Sunny'=> 0.4, 'Stormy'=> 0.1 };

    my $transition = {
       'Rainy' => {'Rainy'=> 0.7, 'Sunny'=> 0.3},
       'Sunny' => {'Rainy'=> 0.4, 'Sunny'=> 0.5, 'Stormy'=>.1},
       };

    my $emission = {
      'shop' => {
                  'Sunny' => '0.4',
                  'Rainy' => '0.9'
                },
      'read' => {
```

```
                'Stormy' => '1'
              },
    'walk' => {
                'Sunny' => '0.6',
                'Rainy' => '0.1'
              },
};

my $v = Algorithm::Viterbi->new();
$v->emission($emission);
$v->transition($transition);
$v->start($start);

my ($prob, $v_path, $v_prob) = $v->forward_viterbi($observations);
  # returns 0, [], 0

$v->{unknown_transition_prob} = 1e-100;

($prob, $v_path, $v_prob) = $v->forward_viterbi($observations);
  # returns 1.62e-102, ['Sunny', 'Sunny', 'Stormy', 'Stormy' ], 4.8e-103;
```

AUTHOR

Koen Dejonghe koen@fietsoverland.com Copyright (c) 2006 Koen Dejonghe. All rights reserved. This program is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

VERSION

Version 0.01 (2006-Nov-07)