‘Supralogic’
or
A method for predicting stochastic mapping outcomes by interpolating their probabilties

Dr. Adam Bourne

✉ a.bourne@physicist.net

Abstract

In a stochastic mapping model, a method is described for interpolating un-sampled mapping probabilities given a successive set of observed mappings. The sampled probabilities are calculated from the observed mappings. The previously described method of interpolating values in code space is used to interpolate the un-observed mapping probabilities. The outcomes for subsequent mappings can then be predicted by finding the processes with maximal interpolated probability.

Finally, a software package is created and demonstrated to implement the method and tested on a variety of situations for filling in missing element values or categorising data arrays.

1 Introduction

This paper takes the result from the previous paper [1] and uses it to explain how to interpolate the probabilities of stochastic mapping processes and so predict their outcomes. This is of use, for example, in classifying strings of symbols, guessing truth table outputs for missing rows, or filling out missing database cell values when the database is incomplete and one wishes to find the most likely possible value.

2 Interpolating values on an undirected graph

2.1 Defining terms

We consider a stochastic mapping, f, which maps the set $M \equiv \{\mathbf{m} = (m_1, m_2, \cdots, m_D)\}$ onto the set $M_0 \equiv \{m_0\}$ where the indices $0 \leq m_j \leq \mu_j \, \forall j : 0 \leq j \leq D$,

$$f : M \rightsquigarrow M_0 \tag{1}$$

and where given mappings have an associated normalised probability function $P_f : M \times M_0 \to [0, 1]$ such that $\sum_{m_0} P_f(\mathbf{m}, m_0) = 1 \, \forall \, \mathbf{m}$. Suppose a set of a total of N_s stochastic mappings has been observed $\{\mathbf{m}(\tau) \to m_0(\tau)\}$ where $1 \leq \tau \leq N_\mathrm{s}$. A set of sampled probabilities can then be calculated from these observations,

$$\rho(\mathbf{m}, m_0) = \frac{1}{n(\mathbf{m})} \sum_\tau \delta_{\mathbf{m},\mathbf{m}(\tau)} \, \delta_{m_0,m_0(\tau)} \quad , \quad \forall \, \mathbf{m} : n(\mathbf{m}) \neq 0 \tag{2}$$

where $n(\mathbf{m}) \equiv \sum_\tau \delta_{\mathbf{m},\mathbf{m}(\tau)}$. These probabilities, with $\mathbf{m} : n(\mathbf{m}) \neq 0$ will be considered to be observed or sampled probabilities for the mapping f. The cases of $\mathbf{m}$ where $n(\mathbf{m}) = 0$ will be regarded as unobserved or un-sampled mapping probabilities. This paper will now seek to apply the interpolation equation to give estimated expected probabilities for the unobserved mapping probabilities, given the observed probabilities.

2.2 Application of the interpolation equation

For each of the possible associated values $\{m_0\}$ there is an associated probability either observed by sampling or inferred by interpolation. This necessitates the use of a set of the following independent interpolation equations,

$$\{\Delta\,\rho(\mathbf{m}, m_0) = Q(\mathbf{m}, m_0)\} \quad , \quad \forall\, m_0 : 0 \leq m_0 \leq \mu_0 - 1 \tag{3}$$

2.3 Conservation of probability

Theorem 2.1. *For normalised distributions at the sample points* $\rho(\mathbf{m}(\tau), m_0) : \sum_{m_0} \rho(\mathbf{m}(\tau), m_0) = 1\,\forall\,\tau$ *then solutions to the set of interpolation equations* (3) *have normalised distributions at all the un-sampled points too, i.e.* $\rho(\mathbf{m}, m_0) : \sum_{m_0} \rho(\mathbf{m}, m_0) = 1\,\forall \mathbf{m}$.

Proof. The sampled probability distributions are normalised,

$$\sum_{m_0} \rho(\mathbf{m}, m_0) = 1 \quad \forall\, \mathbf{m} : n(\mathbf{m}) \neq 0 \tag{4}$$

and by the linearity of the interpolation equation,

$$\Delta \sum_{m_0} \rho(\mathbf{m}, m_0) = \sum_{m_0} Q(\mathbf{m}, m_0) \quad \forall\, \mathbf{m} \tag{5}$$

Given that at the sampled points $\mathbf{m} : n(\mathbf{m}) \neq 0$ all the sampled values of $\sum_{m_0} \rho(\mathbf{m}, m_0)$ are unity, then the trivial solution to the interpolation equation (for all sample values being equal) is then,

$$\sum_{m_0} \rho(\mathbf{m}, m_0) = 1 \quad \forall\, \mathbf{m} \tag{6}$$

i.e. *the interpolation equation conserves probability and ensures all the un-sampled distributions* $\rho(\mathbf{m}, m_0)$ *are normalised over* m_0*, for all* $\mathbf{m}$. □

So the solutions to the interpolation equations preserve teh conservation of probability and so represent valid stochastic mappings.

2.4 General solution to the interpolation equations

The N_s observed mappings $\{\mathbf{m}(\tau) \to m_0(\tau)\}$ gave rise to a set of N observed probabilities at the points $\{\mathbf{m}_n\} \equiv \cup_\tau \{\mathbf{m}(\tau)\}$. Applying the general solutions to each of the equations in (3) we write,

$$\left\{ \rho(\mathbf{m}, m_0) = \lambda_{m_0} + \sum_{n=1}^{N} Q(\mathbf{m}_n, m_0)\, g(\{d_\mu(\mathbf{m}, \mathbf{m}_n)\}) \right\} \tag{7}$$

The pseudo-charges are determined by the set of linear equations,

$$\left\{ \begin{pmatrix} \mathbf{r}(m_0) \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{g} & \mathbf{1} \\ \mathbf{1}^\mathrm{T} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{q}(m_0) \\ \lambda(m_0) \end{pmatrix} \right\} \tag{8}$$

where,

$$\mathbf{r}(m_0) = \begin{pmatrix} \rho(\mathbf{m}_1, m_0) \\ \rho(\mathbf{m}_2, m_0) \\ \vdots \\ \rho(\mathbf{m}_N, m_0) \end{pmatrix}, \quad \mathbf{1} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}, \quad \mathbf{q}(m_0) = \begin{pmatrix} Q(\mathbf{m}_1, m_0) \\ Q(\mathbf{m}_2, m_0) \\ \vdots \\ Q(\mathbf{m}_N, m_0) \end{pmatrix} \tag{9}$$

and the rank N, symmetric matrix $\mathbf{g}$ has components $g_{n,n'} = g(\{d_\mu(\mathbf{m}_n, \mathbf{m}_{n'})\})$.

2.5 Most likely interpolated mapping

Given the solution (7), then the most likely interpolated outcome for the mapping $\mathbf{m} \to m_0$ is,

$$m_0 = \arg\max_{m_0'} \rho(\mathbf{m}, m_0') \tag{10}$$

3 Implementation

A demonstration of the method is now presented using JavaScript and HTML, so as to illustrate how easily the technique could be used client-side using limited processing power. The `math.js` module is used to perform, in particular the solving of linear equations required to calculate the pseudo-charges. A wrapper object `PI` (Probability Interpolator) is used as a name space to contain all the necessary helper functions. The main function for interpolating probabilities is `PI._get_prediction(training_arrays, bitmask_array, input_array)`, which is used in the following manner in the `.js` file,

```
var training_arrays = [
    ['B', 'E', 'N', 'N', 'Y', ' '],
    ['B', 'I', 'L', 'L', 'Y', ' '],
    ['A', 'D', 'R', 'I', 'A', 'N'],
    ['A', 'D', 'A', 'M', ' ', ' '],
    ['T', 'H', 'O', 'M', 'A', 'S'],
    ['T', 'O', 'M', 'M', 'Y', ' '],
    ['E', 'D', 'W', 'A', 'R', 'D'],
    ['E', 'D', 'D', 'I', 'E', ' '],
    ['C', 'H', 'A', 'R', 'L', 'Y'],
    ['B', 'A', 'R', 'R', 'I', 'E']
];

var bitmask_array = [null, true, true, null, true, true],
    input_array = ['B', null, null, 'M', null, null];

var prediction = PI._get_prediction(training_arrays, bitmask_array, input_array);
```

where `training_arrays` is a array of training example arrays, `bitmask_array` is an array of booleans (or Truthy/Falsy values). The purpose of `bitmask_array` is to signal which elements have values that are unknown in `input_array`. The purpose of the `PI._get_prediction` function is to produce a data object containing the interpolated probabilities for each of the masked (unsampled) element values given the unmasked (sampled) ones. In the example snippet above, each array has 6 elements labelled with indices beginning at zero, `input_array` has elements `input_array[0] = 'B'` and `input_array[3] = 'M'`, with the other elements unknown and masked i.e. with `bitmask_array[1] = true`, `bitmask_array[2] = true`, `bitmask_array[4] = true` and `bitmask_array[5] = true`. The output for this example gives the object,

```
prediction = {
  "probability_spectra": {
    "1": {
      "A": 0.157214890438247,
      "D": 0.3037622238319449,
      "E": 0.1636578685258964,
      "H": 0.12840399764578048,
      "I": 0.16365786852589637,
      "O": 0.08330315103223468
    },
    "2": {
      "A": 0.20597100235421942,
      "D": 0.05177924664976458,
      "L": 0.16365786852589637,
      "M": 0.08330315103223468,
      "N": 0.1636578685258964,
      "O": 0.08330315103223468,
      "R": 0.19163176838102136,
      "W": 0.05669594349873229
    },
    "4": {
      " ": 0.16087015574067368,
      "A": 0.117720028975009,
      "E": 0.05177924664976458,
      "I": 0.157214890438247,
      "L": 0.045100846613545804,
      "R": 0.05669594349873229,
      "Y": 0.4106188880840275
    },
    "5": {
      " ": 0.6232682904744659,
      "D": 0.05669594349873229,
```

```
      "E": 0.157214890438247,
      "N": 0.03441687794277434,
      "S": 0.08330315103223468,
      "Y": 0.045100846613545804
    }
  },
  "predicted_element_values": {
    "1": "D",
    "2": "A",
    "4": "Y",
    "5": " "
  }
}
```

This shows the interpolated probability spectra of values for the unspecified elements in `prediction.probability_spectra`. Also summarised in `prediction.predicted_element_values` are the most probable values for each of the given unspecified elements. In this case, when these missing values are placed in the masked elements, `input_array` maps like `['B', null, null, 'M', null, ' ']` → `['B', 'D', 'A', 'M', 'Y', ' ']`.

Another example shows how animal categories can be guessed given a few examples. Animals categories are Mammal, Bird, Reptile and Amphibian. Properties of examples: gives birth to eggs/live young, fur/feathers/scales/smooth skin, warm/cold blood, webbed/non-webbed feet, (lives in) water/-land/both, flies/flightless

```
var training_arrays = [
    ['eggs', 'smooth skin', 'cold', 'webbed',     'both', 'flightless', 'Amphibian'],
        // Frog
    ['eggs', 'feathers',    'warm', 'webbed',     'both', 'flies',      'Bird'],
        // Duck
    ['eggs', 'scales',      'cold', 'non-webbed', 'both', 'flightless', 'Reptile'],
        // Crocodile
    ['live', 'fur',         'warm', 'non-webbed', 'land', 'flightless', 'Mammal'],
        // Cat
    ['live', 'smooth skin', 'warm', 'non-webbed', 'land', 'flightless', 'Mammal'],
        // Human
    ['eggs', 'scales',      'cold', 'non-webbed', 'land', 'flightless', 'Reptile'],
        // Snake
    ['live', 'fur',         'warm', 'non-webbed', 'land', 'flightless', 'Mammal'],
        // Squirrel
    ['eggs', 'feathers',    'warm', 'non-webbed', 'land', 'flies',      'Bird'],
        // Sparrow
];

var bitmask_array = [null, null, null, null, null, null, true];
```

where just the final element value (animal category) is to be predicted. Testing this we see what category is guessed for the Platypus,

```
/* Test for row: ['eggs', 'fur', 'warm', 'webbed', 'both', 'flightless', null], //
    Platypus */

var input_array = ['eggs', 'fur', 'warm blood', 'webbed feet', 'both', 'flightless',
    null],
    prediction = PI._get_prediction(training_arrays, bitmask_array, input_array);
```

The output prediction for these properties is Bird with Reptile, Mammal and Amphibian ranking in descending order of likelihood. This is unsurprising since the only property which is not associated with the Bird category is 'fur':

```
prediction = {
  "probability_spectra": {
    "6": {
      "Amphibian": 0.19864148793795078,
      "Bird": 0.32880046026382864,
      "Mammal": 0.23569325493027315,
      "Reptile": 0.23686479686794754
    }
  },
  "predicted_element_values": {
    "6": "Bird"
```

```
  }
}
```

Further test input values are for Bats, which is correctly guessed as Mammal:

```
/* Test for row: ['live', 'fur', 'warm', 'non-webbed', 'land', 'flies', null], // Bat
   */

var input_array = ['live', 'fur', 'warm ', 'non-webbed', 'land', 'flies', null],
    prediction = PI._get_prediction(training_arrays8, bitmask_array, input_array);
```

which gives

```
 prediction = {
  "probability_spectra": {
    "6": {
      "Amphibian": 0.1284247021888966,
      "Bird": 0.31029955867841297,
      "Mammal": 0.31705975278610665,
      "Reptile": 0.24421598634658384
    }
  },
  "predicted_element_values": {
    "6": "Mammal"
  }
}
```

and Ostriches:

```
/* Test for row: ['eggs', 'feathers', 'warm', 'non-webbed', 'land', 'flightless', null
   ], // Ostrich */

var input_array = ['eggs', 'feathers', 'warm', 'non-webbed', 'land', 'flightless',
    null],
    prediction = PI._get_prediction(training_arrays, bitmask_array, input_array);
```

which are estimated to be in the Bird category,

```
prediction = {
  "probability_spectra": {
    "6": {
      "Amphibian": 0.10114239606580813,
      "Bird": 0.37019621562595584,
      "Mammal": 0.2821499730149964,
      "Reptile": 0.24651141529323972
    }
  },
  "predicted_element_values": {
    "6": "Bird"
  }
}
```

4 Conclusion

The method appears to give sensible predictions for the data sets used. For larger data sets, caching needs to be introduced, since many of the discrete functions involved have a finite number of values and so with memory not at a premium, their values can be stored rather than repeating the same calculations.

References

[1] Bourne, A 2019 *Interpolating Values in Code Space* (`http://vixra.org/pdf/1904.0184v5.pdf`).