Mixed-effects inference for classification studies

Introduction

Classification algorithms are often used in a hierarchical setting, where a classifier is trained and tested on individual datasets which are themselves sampled from a group. Examples of this sort of analysis are ubiquitous and are common in domains as varied as spam detection, brain-machine interfaces, and neuroimaging.

This toolbox provides answers to the questions of statistical inference that arise in all of these settings. It implements models that account for both within-subjects (fixed-effects) and between-subjects (random-effects) variance components and thus provide mixed-effects inference.

The software is extremely easy to use and requires no prerequisites other than MATLAB and the MATLAB Statistics Toolbox.

Literature

For details on the theoretical foundation, practical applications, and advantages over alternative methods, see:

- K.H. Brodersen, J. Daunizeau, C. Mathys, J.R. Chumbley, J.M. Buhmann, & K.E. Stephan. Variational Bayesian mixed-effects inference for classification studies. *NeuroImage* (in press). doi:10.1016/j.neuroimage.2013.03.008.

- K.H. Brodersen, C. Mathys, J.R. Chumbley, J. Daunizeau, C.S. Ong, J.M. Buhmann, & K.E. Stephan. Mixed-effects inference on classification performance in hierarchical data sets. *Journal of Machine Learning Research* (2012), 13, 3133-3176.

- K.H. Brodersen, C.S. Ong, J.M. Buhmann, & K.E. Stephan. The balanced accuracy and its posterior distribution. *ICPR* (2010), 3121-3124.

Example 1 – inference on the accuracy

Consider a situation in which a classification algorithm (e.g., a support vector machine or a logistic regression model) has been trained and tested to predict the binary label (+1 or -1) of a set of trials. Further, assume the analysis has been carried out independently for each subject within a group. The results can then be summarized in terms of two vectors: The first one, k, encodes the number of correctly classified trials in each subject; the second, n, encodes the total number of trials in each subject. The following steps outline how to apply the toolbox to this setting.

Step 1: note down observed classification outcomes

We begin by specifying two vectors that fully describe the observed outcomes of our classification analysis:

```
>> ks = [ 82  75  92  85  88];
>> ns = [100 100 100 100 100];
```

This says, for example, that 82 out of 100 trials were classified correctly in the first subject. There are 5 subjects in total in this example.

Step 2: inference

We perform inference by typing:

```
>> [mu,p,ci] = micp_stats(ks,ns)
```

The above code performs full Bayesian inference using an efficient variational Bayes algorithm. The acronym in micp_stats() is short for **m**ixed-effects **i**nference on **c**lassification **p**erformance. The output comprises three variables:

```
mu =
      0.820
p =
      0.000
ci =
      0.720    0.896
```

This result tells us that the posterior mean of the population mean accuracy is 82%. This is our best estimate of the performance of the classifier (under an implicit ℓ_2-loss function). The next number is the posterior infraliminal probability. It represents the posterior probability mass for classification accuracies below chance (50%). The fact that it is approximately 0 means that we are approximately 100% sure that the population mean accuracy is above chance. Finally, the central 95% credible interval is [72.0%, 89.6%]. It represents the interval in which we place 95% of our posterior belief, and we could use it for plotting error bars on the classification performance.

The function micp_stats() provides various other options. To see these, type:

```
>> help micp_stats
```

Example 2 – inference on the balanced accuracy

In many real-world problems, the data used for classification are not perfectly balanced. This means that there are more examples from one class then from the other. Denoting the two classes as the *positive* and the *negative* class, respectively, there might for instance be more positive than negative examples in the data. When the data are imbalanced, the accuracy is a misleading performance measure and should be replaced by the *balanced accuracy*.

To infer on the balanced accuracy, the toolbox needs to know how many positive and negative trials were classified correctly (rather than just an overall number of correctly classified trials, as was sufficient in Example 1).

Step 1: note down observed class-specific classification outcomes

We begin by noting down how many trials were classified correctly in each subject. In contrast to Example 1, we are now providing this information separately for positive and negative examples. Thus, k and n are now matrices. The first row refers to positive examples, the second row to negative examples.

```
>> ks = [40 44 18 42 44; ...
         48 41 65 49 32];
>> ns = [45 51 20 46 48; ...
         55 49 80 54 32];
```

Here, we recorded that in the first subject, there were 45 examples with true label '+1', out of which 40 were classified correctly. 55 examples had a '-1' label, and 48 of these were classified correctly. Note that in the above example the last subject has fewer trials than the rest; mixed-effects inference will correctly account for this.

Step 2: inference

Inference is as straightforward as before. Since k and n are now matrices (as opposed to row vectors as in Example 1), the toolbox automatically switches to an algorithm for inference on the *balanced* accuracy.

```
>> [mu,p,ci] = micp_stats(ks,ns)
mu =
      0.856
p =
      0.000
ci =
      0.793    0.906
```

This tells us that the posterior mean of the population mean balanced accuracy is 85.6%. Is this better than chance? Yes, with a conviction of $1 - 0.000 = 100\%$. If we wanted to plot error bars, we would use the limits of the central 95% credible interval, which is [79.3%, 90.6%].

Example 3 – choosing a model

Statistical inferences are based on models that embody specific assumptions about the data. In the two examples above, micp_stats() automatically picked an appropriate model for us. However, the toolbox actually contains several such models, each of which has different characteristics.

The default models are implemented using a variational Bayes (VB) approach, which is highly efficient but only provides an approximate result. If we wish to check how accurate this approximation is, we can resort to an alternative set of models that are implemented using Markov chain Monte Carlo (MCMC). These can be exceedingly slow but are asymptotically correct in the limit of an infinite runtime.

model	inferred quantity	implementation
univariate normal-binomial model (unb_vb)	accuracy	VB
univariate normal-binomial model (unb_mcmc)	accuracy	MCMC
univariate beta-binomial model (ubb_mcmc)	accuracy	MCMC
twofold normal-binomial (tnb_vb)	balanced accuracy	VB
twofold beta-binomial (tbb_mcmc)	balanced accuracy	MCMC
bivariate normal-binomial model (bnb_mcmc)	balanced accuracy	MCMC

By default, the toolbox picks a VB approach for inference. To use an approach other than this default, specify the model using its five-letter acronym. For example, if we wanted to rerun the analysis in Example 2 on the basis of an MCMC implementation under the exact same distributional assumptions, we would type:

```
>> [mu,p,ci] = micp_stats(ks,ns,'model','unb_mcmc')
```

Note that, since MCMC implementations are stochastic, the output will vary slightly each time we run the function; however, it stabilizes as we increase the number of samples:

```
>> [mu,p,ci] = micp_stats(ks,ns,'model','unb_mcmc','nSamples',1e6)
```

Example 4 – deciding between alternative models

As shown in the table above, there are two models both of which provide inference on balanced accuracies using MCMC (tbb_mcmc and bnb_mcmc). Which model should we use for inference? The answer can be obtained by Bayesian model comparison:

```
>> logBF = bicp_bms(ks,ns)

Bayesian model comparison:
log BF = 3.4054
There is 'strong' evidence in favour of the twofold beta-binomial (tbb) model.
```

The above result tells us that, in this case, the twofold beta-binomial provides a much better explanation of the data than the bivariate normal-binomial model. We therefore proceed with inference using the twofold beta-binomial model:

```
[mu,p,ci] = micp_stats(ks,ns,'model','tbb_mcmc')
mu =
     0.871
p =
     0.000
ci =
     0.799    0.917
```

Reassuringly, compared to our results in Example 2, the implications of the switch in model on our conclusions have been moderate. Note that model comparison of this sort is currently only implemented for comparing tbb_mcmc and bnb_mcmc.

Example 5: visualizing results

The toolbox includes a demo script that performs inferences and visualizes the results. The first demo illustrates the use of the default models for inference on accuracies and balanced accuracies. These models are based on VB implementations. To run the demo, type:
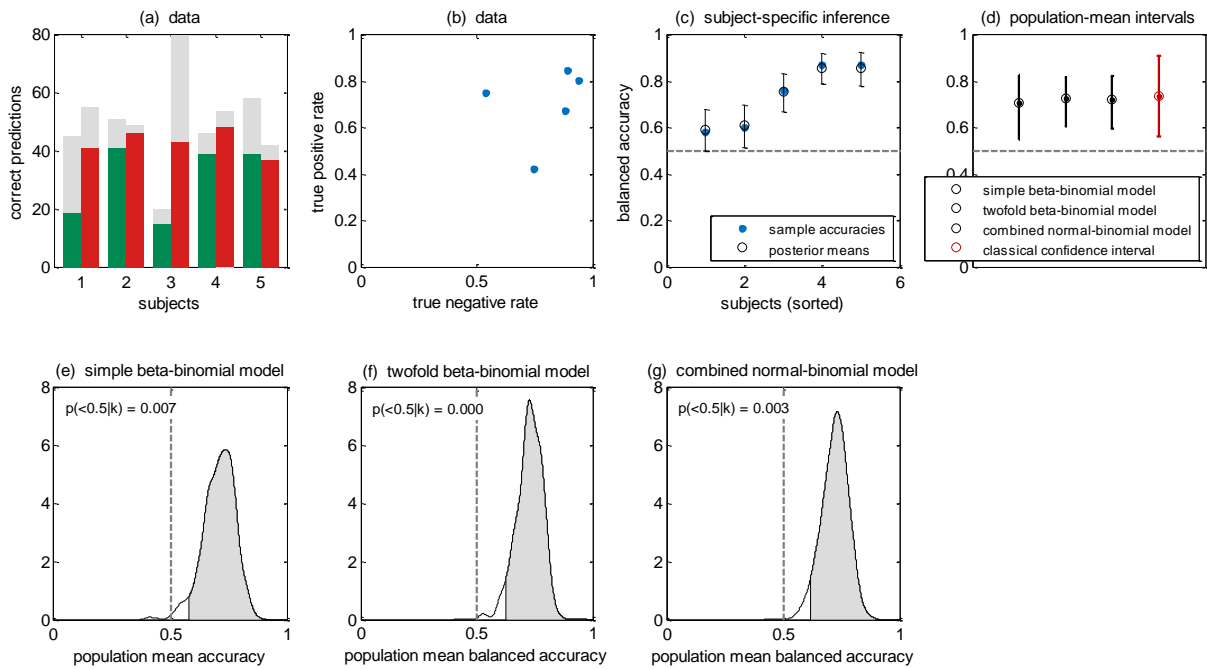
```
>> micp_demo_vb
```

The demo produces the following figure in a step-by-step fashion:
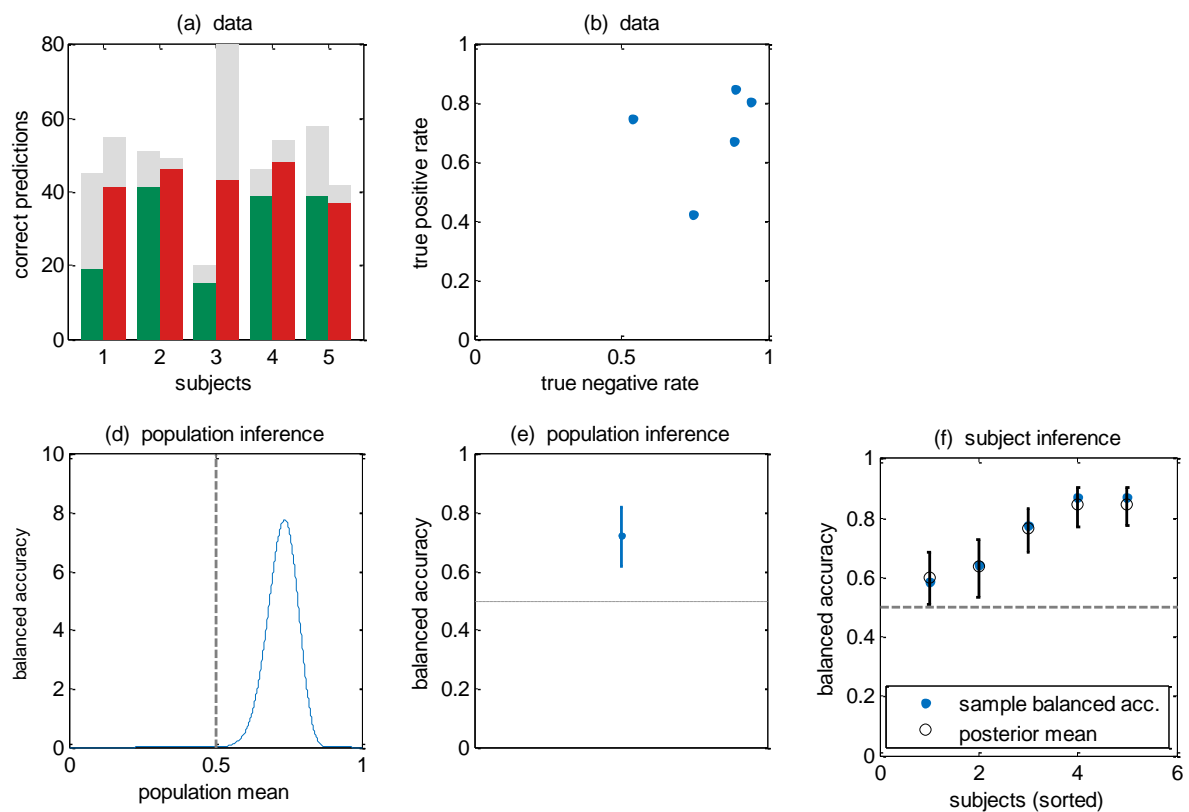
The second demo provides an example of how to visualize inferences obtained through MCMC-based models. To run this demo, type:

```
>> micp_demo_mcmc
```

The demo produces the figure shown below:

Summary of toolbox contents

The primary interface for the toolbox is the function micp_stats(), as described throughout this manual. The list below provides an overview of other key functions contained in this toolbox.

Variational inference algorithms

- vbicp_unb   Variational approximate inference using the univariate normal-binomial model.

Sampling algorithms

- bicp_sample_unb   Computes accuracy samples using the univariate normal-binomial model.
- bicp_sample_ubb   Computes accuracy samples using the univariate beta-binomial model.
- bicp_sample_ubb_par   Parallel version for use with the MATLAB Distributed Computing Toolbox.
- bicp_sample_bnb   Computes balanced-accuracy samples using the bivariate normal-binomial model.
- bicp_sample_bnb_par   Parallel version for use with the MATLAB Distributed Computing Toolbox.

Model comparison

- bicp_evidence_bb   Computes the log model evidence of the beta-binomial model.
- bicp_evidence_tnb   Computes the log model evidence of the twofold beta-binomial model.
- bicp_evidence_bnb   Computes the log model evidence of the bivariate normal-binomial model.
- bicp_bms   Performs Bayesian model comparison on the tbb and bnb models.

Example

- micp_demo.m   Generates synthetic data, performs inference, and plots the results.

Software note

Kay H. Brodersen
ETH Zurich
Switzerland
brodersen@biomed.ee.ethz.ch