



The Decoding Toolbox (TDT): a versatile software package for multivariate analyses of functional imaging data

Martin N. Hebart^{1,2,3,4,*†}, Kai Gørgen^{2,3,5,*†} and John-Dylan Haynes^{2,3,4}

¹ Department of Systems Neuroscience, University Medical Center Hamburg-Eppendorf, Hamburg, Germany

² Bernstein Center for Computational Neuroscience, Charité Universitätsmedizin, Berlin, Germany

³ Berlin Center for Advanced Neuroimaging, Charité Universitätsmedizin, Berlin, Germany

⁴ Berlin School of Mind and Brain, Humboldt-Universität zu Berlin, Berlin, Germany

⁵ Fachgebiet Neurotechnologie, Technische Universität Berlin, Berlin, Germany

Edited by:

Arjen Van Ooyen, VU University
Amsterdam, Netherlands

Reviewed by:

Julien Dubois, California Institute of
Technology, USA

Qiyong Gong, West China Hospital
of Sichuan University, China

*Correspondence:

Martin N. Hebart, Department of
Systems Neuroscience, University
Medical Center

Hamburg-Eppendorf, W34,
Martinistraße 52, 20251 Hamburg,
Germany

e-mail: m.hebart@uke.de;

Kai Gørgen, Bernstein Center for
Computational Neuroscience,
Charité Universitätsmedizin, Haus 6,
Philippstr. 13, 10115 Berlin, Germany
e-mail: kai.goergen@bccn-berlin.de

[†] These authors have contributed
equally to this work.

The multivariate analysis of brain signals has recently sparked a great amount of interest, yet accessible and versatile tools to carry out decoding analyses are scarce. Here we introduce *The Decoding Toolbox* (TDT) which represents a user-friendly, powerful and flexible package for multivariate analysis of functional brain imaging data. TDT is written in Matlab and equipped with an interface to the widely used brain data analysis package SPM. The toolbox allows running fast whole-brain analyses, region-of-interest analyses and searchlight analyses, using machine learning classifiers, pattern correlation analysis, or representational similarity analysis. It offers automatic creation and visualization of diverse cross-validation schemes, feature scaling, nested parameter selection, a variety of feature selection methods, multiclass capabilities, and pattern reconstruction from classifier weights. While basic users can implement a generic analysis in one line of code, advanced users can extend the toolbox to their needs or exploit the structure to combine it with external high-performance classification toolboxes. The toolbox comes with an example data set which can be used to try out the various analysis methods. Taken together, TDT offers a promising option for researchers who want to employ multivariate analyses of brain activity patterns.

Keywords: multivariate pattern analysis, decoding, pattern classification, fMRI, representational similarity analysis, searchlight

INTRODUCTION

Human neuroscientists are interested in understanding the function of the human brain and nervous system. For that purpose, they have developed numerous methods that directly or indirectly measure the activity of the nervous system at work. One of these methods is functional magnetic resonance imaging (fMRI) which measures brain activity indirectly through the blood oxygen level dependent (BOLD) response (Logothetis et al., 2001). Conventionally, the focus of fMRI has been to perform mass-univariate analyses, i.e., to analyze the recorded data time courses of each fMRI brain voxel (for MEG/EEG each sensor/electrode) separately, for example with the general linear model (GLM, e.g., Friston et al., 1994).

More recently, these mass-univariate analysis methods have been complemented by *multivariate pattern analysis* (MVPA) which refers to a collection of multivariate methods of brain data analysis that incorporate multiple dependent variables at the same time (Haynes and Rees, 2006; Norman et al., 2006; Kriegeskorte, 2011). One of the most popular multivariate methods of brain data analysis is generally referred to as multivariate *decoding* which describes the mapping of multiple dependent variables to one or multiple independent variables, and which contrasts with multivariate *encoding* describing the opposite mapping (Naselaris

et al., 2011). The popularity of multivariate methods of brain data analysis stems from three facts: First, multivariate methods offer increased *sensitivity* in detecting statistical dependence between cognitive variables and (patterns of) brain activity, because these methods can combine information across multiple voxels and exploit their covariance. Put simply, multivariate methods make it easier to detect existing differences between brain signals. Second, multivariate methods allow for greater *specificity* in finding a statistical dependence between measured brain data and multiple categorical responses. Although a number of brain regions show overall changes in activity selective for specific stimulus categories (e.g., Kanwisher et al., 1997), stimuli can be encoded in a more distributed manner (Averbeck et al., 2006). Multivariate methods can be used to target distributed representations of stimuli, cognitive variables, or other variables of interest. This effectively enables researchers to ask more specific questions regarding neuronal representations, for example about the representation of individual face exemplars (Kriegeskorte et al., 2007; Nestor et al., 2011). Third, multivariate decoding methods allow functional neuroimaging to be used for accurate prediction of cognitive and mental states in the form of a “neuronal marker.” For example, they might be used to support the diagnosis of neurological disorders beyond regions typically detected by radiologists (Weygandt

et al., 2011), predict the conversion from mild cognitive impairment to Alzheimer's disease (Cui et al., 2011), interrogate cognitive states from people that cannot communicate (Horikawa et al., 2013), or support non-invasive brain-computer interfaces for motor control in paralyzed patients (Blankertz et al., 2007).

Although the number of publications using multivariate decoding methods has been rising continuously, it still seems to be common for researchers to program their own multivariate data analysis pipeline from scratch¹. This is not only very time-consuming and redundant, it is also very error-prone. Indeed, if the same piece of code is adapted for different decoding analyses, changes that were done for previous analyses might be overlooked in later analyses or might affect other analysis steps which can produce invalid results. Just as problematic—and not uncommon even for experts—is the unintentional wrong use of decoding methods, for example circular analyses (Kriegeskorte et al., 2009). In addition, researchers might hesitate to conduct decoding analyses, either because of their limited understanding of machine learning methods for functional neuroimaging (covered in detail by Formisano et al., 2008; Pereira et al., 2009; Lemm et al., 2011) or because of the effort involved in learning to apply a new analysis method. For that reason, researchers getting started with decoding would strongly benefit from an easy-to-use implementation of multivariate methods. At the same time, they would ideally keep using the same software for simple analyses that they would later use for more sophisticated approaches once they have gained sufficient knowledge.

Here we present *The Decoding Toolbox* (TDT) that offers a flexible yet easy to use framework for decoding analyses in Matlab (Mathworks, Natick, MA). TDT offers an interface with SPM (Statistical Parametric Mapping, <http://www.fil.ion.ucl.ac.uk/spm/>), the most common fMRI data analysis software, but can also be extended to other fMRI data analysis packages. TDT can be used by any researcher with basic knowledge in fMRI data analysis and minimal programming skills, while at the same time allowing a large amount of flexibility in the application of complex decoding approaches for more experienced researchers. We briefly summarize the key features of the toolbox that should make its use attractive.

SIMPLICITY

Originally, we created TDT to simplify setting up decoding analyses and prevent the unintentional introduction of programming errors or unnoticed changes in settings that do not elicit error

¹To support this observation numerically, we conducted a Google Scholar search where we intended to limit our search to articles using multivariate decoding methods and which explicitly mention the software package they use for regular analyses. These articles should also mention the MVPA toolbox used. For the search, we used the combination “fMRI AND (‘multivariate pattern analysis’ OR ‘multivoxel pattern analysis’) AND (FSL OR AFNI OR Freesurfer OR SPM).” This yielded 659 hits between 2010 and 2013. Of these, only 103 (15.6 %) contained the strings “PRoNTTo,” “PyMVPA,” “MVPA toolbox” or “3dsvm,” which refer to previous MVPA-related software packages (see Comparison to Existing Packages of Multivariate Data Analysis). 132 (20.0 %) contained the terms “in-house,” “custom-written,” “custom-made,” “custom-built” or “custom Matlab.” Neither term was contained in 442 (67.0%) articles.

messages, but which may severely compromise results. For that reason, we wanted to make TDT particularly easy to use, where all important settings are made at the beginning. We believe that minimal programming ability is necessary, with the only requirement being that a user is able to modify existing Matlab scripts that define the settings for the decoding analysis. This simplicity also allows users to see at a glance what decoding analysis they are performing.

MODULAR STRUCTURE AND TRANSPARENCY

The modular structure and use of encapsulation for different sub-routines make the toolbox relatively easy to understand. At the same time, this structure is less error prone and reduces programming effort from the side of developers. For users, the existence of decoding design matrices (see below) and the possibility of extensive logging make the internal processes of the toolbox more transparent, which also reduces the probability of mistakes.

SPM INTERFACE

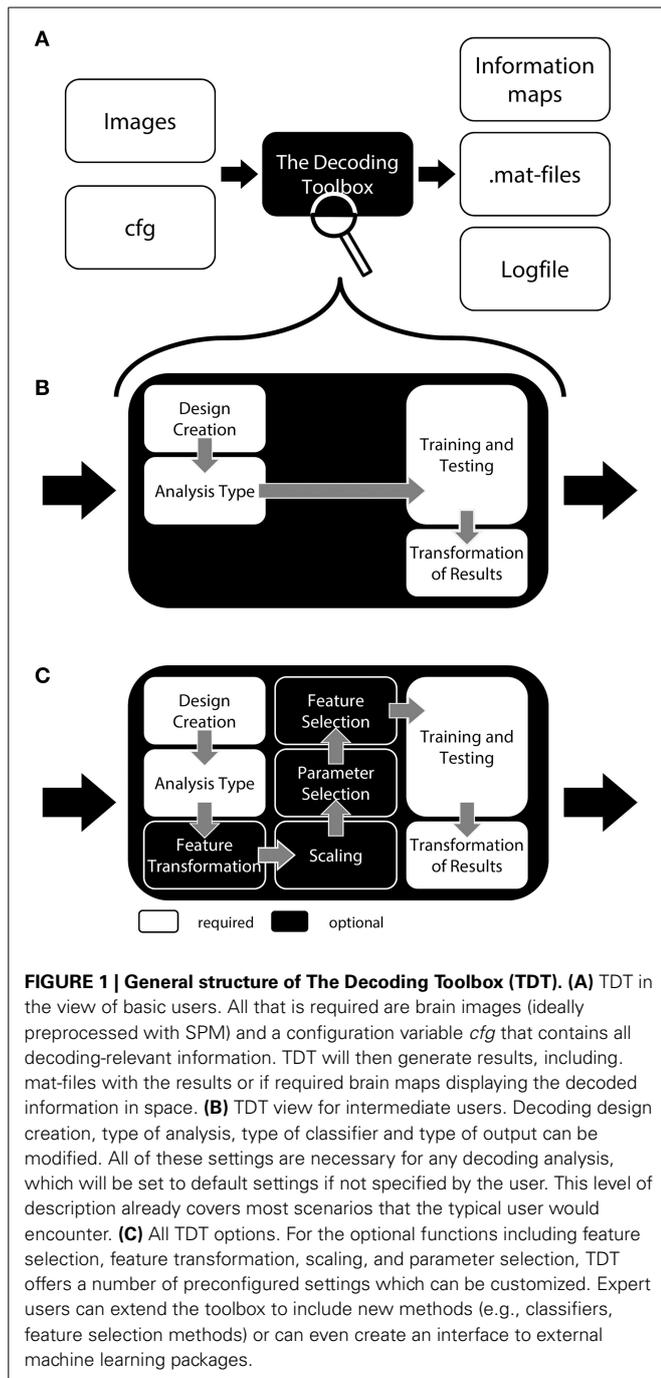
TDT has been developed from an SPM perspective and works with current SPM versions (downwards compatible to SPM2), but can also be used outside of SPM. TDT can directly read all decoding-relevant information from SPM design matrices and in that way simplifies the process of setting up standard decoding analyses. For example, a complete leave-one-run-out cross-validation decoding analysis can be executed using one line of code, by providing the path to the analysis and the names of the regressors that should be classified (see below).

MODULARITY, VERSATILITY, EXTENSIBILITY

TDT comes with a number of methods which allow for executing most commonly used analysis pipelines “out of the box” (Figure 1). These include feature scaling, feature transformation (e.g., principal component analysis), parameter selection, and feature selection as steps prior to decoding. The decoding analysis itself has two-class and multiclass classification implemented, using a number of often-used classifiers including support vector classifiers, logistic regression, and correlation classifiers (Haxby et al., 2001). Additionally, support vector regression is part of the toolbox. The toolbox has been tested and runs under Windows, Linux, and Mac OS.

Due to the modular structure and the simplicity of the code, users can extend TDT with little effort to incorporate other methods as well. Other software packages can be used instead of SPM, new classifiers can be introduced, other means of feature selection can be applied, and even complete toolboxes—e.g., other Matlab decoding toolboxes—can be interfaced to run in the TDT framework.

We chose to implement our toolbox in Matlab because it is a very popular high-level programming language in the neuroimaging community and offers an excellent programming environment with a large and helpful community. We are aware that part of the neuroimaging community is trying to move toward completely non-commercial software instead. TDT does, however, not require any additional commercial Matlab toolboxes such as the Statistics or Bioinformatics Toolbox. Many functions were written specifically for TDT, for example calculation



of *F*-values, implementation of statistical distributions, and quick 1-D linear interpolation.

SPEED

TDT has been profiled extensively to identify and remove possible bottlenecks that would otherwise reduce the speed of decoding analyses. For example, TDT can achieve a three-fold speed increase in cross-validation designs by computing a kernel (Müller et al., 2001) only once and then passing the training and test part of that kernel to the classifier. The resulting speed improvement becomes particularly apparent with large

classification problems or searchlight analyses. In addition, we rewrote many built-in Matlab functions, for example for creating correlation matrices or finding unique values, considerably speeding-up processing.

STRONG DEBUGGING AND ERROR CHECKING

We wanted to make sure that there are as few programming errors as possible in the code, a goal that is particularly difficult given the flexibility and diversity of possible decoding analyses. Although in general it is never possible to guarantee that any software is free from bugs, we regularly run a number of basic analyses as benchmarks to check if these analyses still run and create the same output as in previous analyses. TDT has been extensively tested by many beta users and, importantly, several published results used prior versions of the toolbox (Hesselmann et al., 2011; Christophel et al., 2012, in press; Hebart et al., 2012, in press; Reverberi et al., 2012; Christophel and Haynes, 2014; Ritter et al., 2014; Van Kemenade et al., 2014a,b; Ludwig et al., 2015; Bilalić et al., in press; Guggenmos et al., in press).

In addition, TDT contains many checks to prevent erroneous analyses. In general, it is always possible that unwanted errors occur, and of course researchers always have to double-check if their analysis yields meaningful results. The implemented checks facilitate the process of detecting and eliminating errors. Among others, they ensure independence of training and test data, that all data has the same realignment parameters, and more generally whether the selected options are incomplete or would generate errors. Other initial checks help in saving time and frustration of the user: Each decoding analysis creates a log-file which can be used to recover possible sources of errors. These features have been implemented based on the experience of the authors as well as through user experience and will continuously be improved.

READABILITY

TDT is open-source (see License Statement), with extensively commented code, comprehensive variable names, and an execution structure that follows along a main function with several auxiliary functions. We did not want to provide a black-box where a user cannot follow the steps that are carried out in a decoding analysis.

In the following we will describe TDT in more detail. We will use the style of a tutorial which is intended to ease readability of this section. We will introduce TDT at three different levels of complexity, starting with a rough description of the toolbox to provide an initial overview of the basic structure of TDT. This section should be sufficient for users just getting started with decoding, or those who would like to conduct a simple generic decoding analysis. Second, we give a more detailed explanation of how to expand on this standard decoding analysis for users at an intermediate level who want to tailor the analysis to their needs. Third, for advanced users we expand on more detailed optional settings of the toolbox, including parameter selection or feature selection. After these three steps, we will briefly demonstrate how users who want to exploit the full capabilities of the toolbox can extend the toolbox to new classifiers, new feature selection methods, or even complete machine learning packages.

We may use terminology unfamiliar to some readers. Rather than explaining all terms in the text, we summarize the most important ones in **Table 1** (for a detailed explanation of classification-related terms, please consult Pereira et al., 2009). Although we provide examples written for users who want to carry out within-subject classification, most of these examples also hold for between-subject classification.

Table 1 | Important terminology for multivariate decoding with The Decoding Toolbox.

Term	Description
Beta maps	After estimating the <i>general linear model</i> , each regressor in each voxel receives a parameter estimate <i>beta</i> which can be used for classification and reflects the fit of the model to the data in form of a regression coefficient. Since each voxel is analyzed separately, brain images of betas (i.e., <i>beta maps</i>) are created for each regressor
Chunk	A unit that determines which data should remain together within a <i>decoding step</i> (often: <i>cross-validation</i> fold). Typically, <i>chunks</i> are used to assign run numbers to data, which in <i>cross-validation</i> should ensure that data is independent and not biased by temporal autocorrelation. For a typical <i>leave-one-run-out cross-validation</i> , there are as many <i>chunks</i> as there are <i>decoding steps</i> . For methods where data samples are drawn repeatedly, the number of <i>decoding steps</i> is often a lot larger than the number of <i>chunks</i>
Cross-classification	Not to be confused with <i>cross-validation</i> . A method to test the stability of a classifier, which can be used (1) to demonstrate the generality of a classifier, (2) for a representational association between two cognitive functions, or (3) simply to show representational stability across time
Cross-validation	Method to estimate of how well a classifier generalizes to novel data. In <i>leave-one-run-out cross-validation</i> , data is partitioned in n_{runs} <i>chunks</i> , and on each <i>decoding step</i> (also called <i>fold</i> or iteration) one <i>chunk</i> is used as test data to evaluate a classifier trained on all other chunks. Typically, the average performance of all <i>decoding steps</i> in terms of classification accuracy is then used to evaluate the classifier. <i>Nested cross-validation</i> is cross-validation that is done only on training data to optimize the performance of the classifier (e.g., in <i>feature</i> or <i>parameter selection</i>)
Curse of dimensionality	Within machine learning the fact that classifier performance, i.e., the predictive power of a classifier drops when the number of features (e.g., voxels) becomes much larger than the number of samples (e.g., brain images)
Decoding step	An iteration of a decoding analysis which is part of the cycle of evaluating the classifier. When <i>cross-validation</i> is used, <i>decoding step</i> refers to a cross-validation fold
Feature selection	Methods that reduce the number of features (e.g., voxels). In our terminology, selecting <i>regions of interest</i> or running a <i>searchlight analysis</i> are not parts of <i>feature selection</i> . In addition, we treat methods separately that reduce the dimensionality, but give up the voxel-to-voxel mapping (e.g., PCA). Within TDT, we refer to methods that change the voxel-to-voxel mapping as <i>feature transformation</i> methods
General linear model (GLM)	A statistical model that incorporates analysis of variance, linear regression, and related parametric tests into a common framework. In brain imaging, the GLM is commonly used to explain each voxel's time series separately with multiple linear regressors each representing conditions of interest or nuisance variables (Friston et al., 1994). The term <i>mass-univariate</i> refers to the fact that the GLM is calculated for each voxel individually
Hyperplane	A plane in more than 3D. Typically, the term <i>separating hyperplane</i> is used which separates the space spanned by different features (e.g., voxels) in two subspaces and defines the decision boundary of the classifier. Each part of the voxel space is in this way assigned to one of two classes
L1/L2-norm	A regularization method which influences the complexity of a classification model. In most cases, an L2-norm is used which minimizes length of the <i>weight vector</i> . The L1-norm can be used to minimize the sum of the absolute weights, typically resulting in a sparser model (i.e., less features will contribute to the classification)
Margin	For <i>support vector machines</i> , the area of space between two classes, of which the center is typically the separating hyperplane. SVMs have the goal of maximizing the margin between two classes
Searchlight analysis	One of the three most common types of decoding analysis conducted, the two other being <i>whole brain decoding</i> and <i>region-of-interest decoding</i> . <i>Searchlight decoding</i> typically creates a map of classification accuracies that can be interpreted as the local information content around each voxel (Kriegeskorte et al., 2006; Haynes et al., 2007)
Support vector machine	A type of classifier that maximizes the <i>margin</i> between two different classes (Cortes and Vapnik, 1995)
Weight vector	Determines the contribution of each feature to the final classifier function. For most classifiers, the weight vector cannot be directly interpreted as reflecting the classified variable, because it is a filter that extracts a class signal while at the same time it suppresses correlated noise. Using the covariance of the data, the weight vector can be converted into an interpretable <i>pattern vector</i> (Haufe et al., 2014) which in case of voxel features can be mapped back to a brain image

A SIMPLE DECODING ANALYSIS

For a general decoding analysis, let us for the moment treat TDT as a black box (**Figure 1A**). The user feeds in a set of brain images that he wants to classify and receives a single classification accuracy or an information map as output. What is needed in addition is the configuration variable *cfg* which carries all information necessary to conduct a specific decoding analysis. Most parameters in *cfg* are defaults that are set automatically; parameters are only set manually when they should be changed. In addition, the SPM-interface can automatically extract the decoding-relevant information from the SPM.mat file which is generated as part of a standard general linear model (GLM) in SPM. This information is then automatically added to *cfg*. This interface can be used if users classify not individual brain images, but use parameter estimates of single trials or runs generated by SPM (e.g., *beta maps*, see **Table 1**). Assuming that a user created the SPM.mat file with unique names for all regressors (e.g., “left” and “right” for regressors related to button presses) and would like to carry out a “standard” leave-one-run-out cross validation scheme with two categories, a complete decoding analysis in TDT can be executed in only one line of code (we will explain the meaning of this below). In short, the example call

```
decoding_example('searchlight', 'left',
                'right', beta_dir, output_dir, 4)
```

will perform a cross-validated leave-one-run-out searchlight decoding analysis between the regressors “left” and “right,” where *beta_dir* is the path where the SPM GLM results are stored, *output_dir* is the folder where the decoding results will be saved, and 4 is the radius of a spherical searchlight in voxels (see next paragraph for a detailed explanation). This analysis will yield a map of accuracies that can be inspected with SPM or other third-party software and which should uncover brain regions involved in encoding the motor response.

To allow a first look inside the “black box,” we will now explain the inner functions of this example call (**Figure 1**). Prior to running a decoding analysis, TDT assumes that data have been preprocessed appropriately (**Figure 2** top). Standard preprocessing includes spatial realignment, possibly slice timing correction and detrending, and in some cases also spatial normalization and smoothing (although these latter steps are less common for preprocessing of decoding analyses). As mentioned above, rather than running a decoding analysis on individual images, it has become quite common to use single trial estimates of data (Mumford et al., 2012) or even estimates that combine multiple trials within one run (Haynes et al., 2007). This process can improve classifier performance (Mourão-Miranda et al., 2006, 2007; Mumford et al., 2012), and combining multiple trials can lead to higher classification accuracies (Ku et al., 2008) and slightly improved power (Allefeld and Haynes, 2014). These estimates would typically be generated in the common GLM framework and are stored as so called beta images. For simplicity, in the following we will assume that one beta image per condition per run is generated (e.g., one for “left” for run 1, one for “right” for run 1, one for “left” for run 2, one for “right” for run 2, etc.). In this case, the above example call simply gets the beta images from *beta_dir* representing the conditions

“left” and “right” and will extract the run numbers. The brain mask which SPM automatically creates during model estimation contained in *beta_dir* is automatically used to reduce the analysis to voxels inside the brain. This information is sufficient to carry out a decoding analysis with a leave-one-run-out cross-validation scheme (see **Table 1** for terminology), and in this example, a so-called searchlight analysis is executed with a radius of 4 voxels, using a support vector machine (SVM, Cortes and Vapnik, 1995) as a classifier. On a Dell Precision M4600 (Intel Core i7-2820QM @ 2.30 GHz, 64 bit Windows 7) with Matlab (2011a), this analysis takes roughly 3 min for around 120,000 searchlights.

For a group analysis, this procedure is repeated for every participant, and the resulting accuracy maps can then be spatially normalized and submitted to standard statistical analysis procedures in SPM or any other preferred software package (**Figure 2** bottom).

A CLOSER LOOK AT THE DECODING TOOLBOX—FOR INTERMEDIATE LEVEL USERS AND ABOVE

Although the most basic one-line-of-code use of the toolbox already allows running a multitude of analyses with practically no programming skills, users may want to adjust a number of settings. In this section, we explain the major steps that are required in each decoding analysis (**Figure 1B**), and in the next section optional steps are explained. For each of these required steps there are default settings in TDT, but the user may wish to adjust them. In general, each user creates a short script which contains (1) all the settings of the *cfg* variable and (2) possibly the automatic extraction of data from an existing SPM model.

Basic structure:

```
cfg = decoding_defaults(); % optional:
    initializes cfg with default values
    ... % additional lines that modify cfg
    might go here
results = decoding(cfg); % Performs
    decoding
```

Concrete example call:

```
cfg = decoding_defaults();
regressor_names = design_from_spm
    (beta_dir); % beta_dir is directory
    of SPM model
cfg = decoding_describe_data
    (cfg, {'left', 'right'}, [1 -1], ...
    regressor_names, beta_dir);
% 1 -1 are arbitrary label numbers
    for red & green
cfg.design = make_design_cv(cfg);
results = decoding(cfg);
```

This example automatically extracts regressor names from the SPM model, then accumulates all relevant information from the design related to the regressors of interest (“left” and “right”), creates a leave-one-run-out cross-validation design, and executes the decoding analysis using default settings. Important basic settings that otherwise use these default settings are explained in the next four paragraphs.

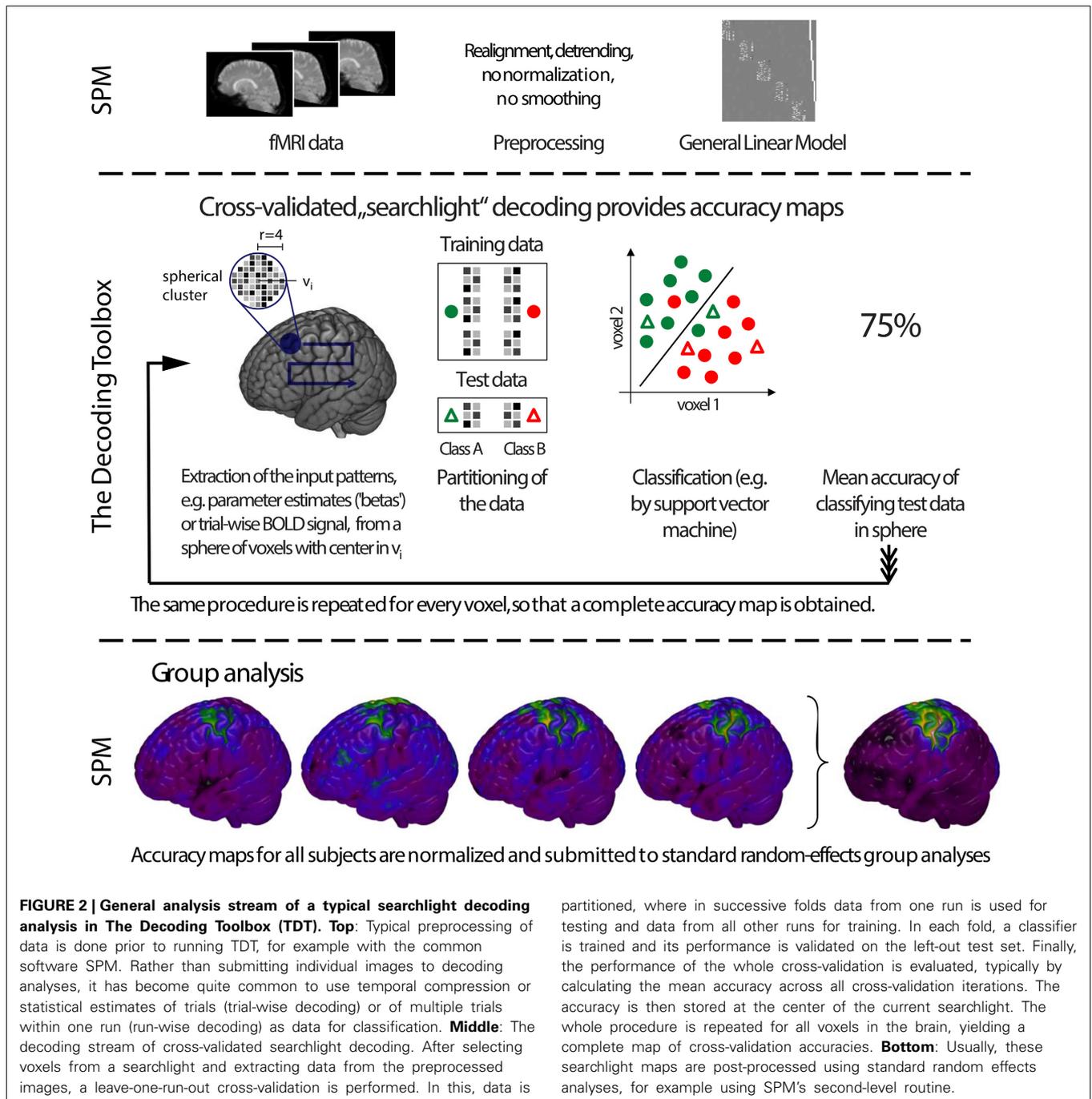


FIGURE 2 | General analysis stream of a typical searchlight decoding analysis in The Decoding Toolbox (TDT). **Top:** Typical preprocessing of data is done prior to running TDT, for example with the common software SPM. Rather than submitting individual images to decoding analyses, it has become quite common to use temporal compression or statistical estimates of trials (trial-wise decoding) or of multiple trials within one run (run-wise decoding) as data for classification. **Middle:** The decoding stream of cross-validated searchlight decoding. After selecting voxels from a searchlight and extracting data from the preprocessed images, a leave-one-run-out cross-validation is performed. In this, data is

partitioned, where in successive folds data from one run is used for testing and data from all other runs for training. In each fold, a classifier is trained and its performance is validated on the left-out test set. Finally, the performance of the whole cross-validation is evaluated, typically by calculating the mean accuracy across all cross-validation iterations. The accuracy is then stored at the center of the current searchlight. The whole procedure is repeated for all voxels in the brain, yielding a complete map of cross-validation accuracies. **Bottom:** Usually, these searchlight maps are post-processed using standard random effects analyses, for example using SPM's second-level routine.

DESIGN CREATION

A powerful feature of TDT is the use of decoding design matrices (Figure 3). A decoding design matrix determines the exact structure of a cross-validation design, i.e., which data belongs to which class and which data is training and test data in each decoding step (also called cross-validation fold or iteration). To determine which samples should remain together (e.g., because they are in one run and the analysis is a leave-one-run-out cross-validation), data can be assigned to chunks which serve as categorical numerical labels. Although in many cases, the number of chunks determines the number of decoding steps, more complex designs

(e.g., bootstrapping designs, Figure 3C) can have more steps than chunks. Finally, multiple decoding analyses can be carried out in one design, a feature which might be particularly useful if data needs to be analyzed separately and combined, or if cross-classification is applied to a number of different data sets saving repeated training on the same data. A number of existing functions allow creating a multitude of different designs in advance, and additional design functions can be created by more advanced users (see Figure 3 for some example designs using different functions). After creating the design, the design inspection can be used to visualize if the analysis is carried out in the correct manner.

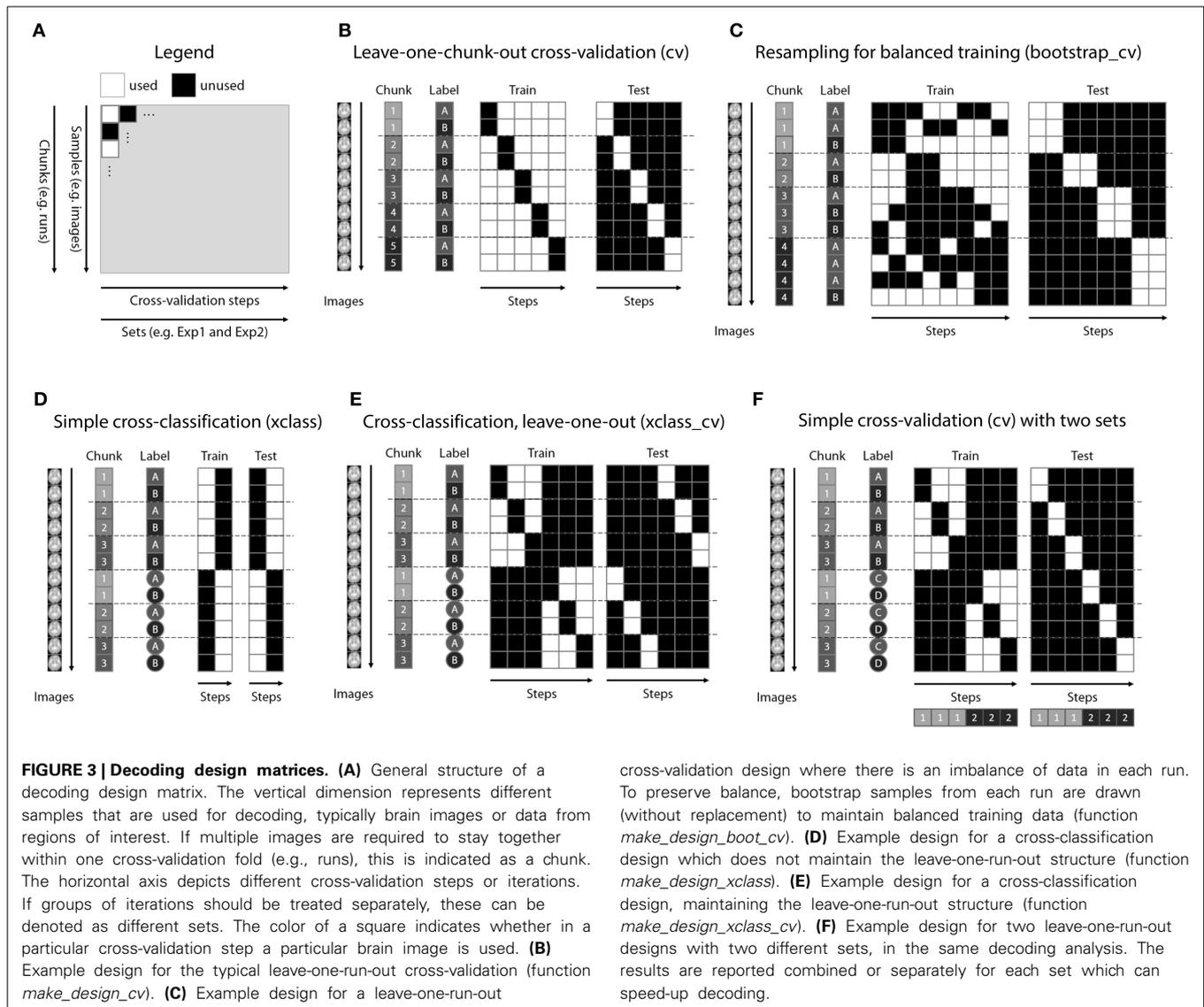


FIGURE 3 | Decoding design matrices. (A) General structure of a decoding design matrix. The vertical dimension represents different samples that are used for decoding, typically brain images or data from regions of interest. If multiple images are required to stay together within one cross-validation fold (e.g., runs), this is indicated as a chunk. The horizontal axis depicts different cross-validation steps or iterations. If groups of iterations should be treated separately, these can be denoted as different sets. The color of a square indicates whether in a particular cross-validation step a particular brain image is used. (B) Example design for the typical leave-one-run-out cross-validation (function *make_design_cv*). (C) Example design for a leave-one-run-out

cross-validation design where there is an imbalance of data in each run. To preserve balance, bootstrap samples from each run are drawn (without replacement) to maintain balanced training data (function *make_design_boot_cv*). (D) Example design for a cross-classification design which does not maintain the leave-one-run-out structure (function *make_design_xclass*). (E) Example design for a cross-classification design, maintaining the leave-one-run-out structure (function *make_design_xclass_cv*). (F) Example design for two leave-one-run-out designs with two different sets, in the same decoding analysis. The results are reported combined or separately for each set which can speed-up decoding.

Example call (design created and visualized prior to a decoding analysis as part of the script):

```
cfg.design = make_design_cv(cfg);
% creates the design prior to decoding
display_design(cfg)% visualizes the
design prior to decoding
```

Alternative example call (design created and visualized while the decoding analysis is running):

```
cfg.design.function.name = 'make_design_cv';
% creates a leave-one-chunk-out
cross-validation design
cfg.plot_design = 1;
% plots the design when decoding starts
```

ANALYSIS TYPE

Most decoding analyses can be categorized as one of three different types of analysis, depending on the general voxel

selection criterion: Whole-brain analyses, region-of-interest (ROI) analyses, and searchlight analyses. All of these approaches are commonly used for decoding. In the machine learning community, ROI and searchlight analyses might be considered feature selection approaches, but this is not typically how they are called in the brain imaging community. All of these approaches have their respective advantages and disadvantages (discussed e.g., in Etzel et al., 2009, 2013). Among others, one advantage of whole-brain analyses is that all available information is fed into the classifier, while major disadvantages of this method are the difficulty to tell the origin of the information and the so-called “curse of dimensionality” (see Table 1). These problems are less of an issue in ROI analyses, but ROIs need to be specified and can in that way be biased, variable ROI sizes make them difficult to compare, and information encoded across different ROIs gets lost. Searchlight analyses can be seen as a succession of spherical ROIs, but require no prior selection of brain regions and are in that respect unbiased to prior assumptions about where to expect an

effect (Friston et al., 2006); however, they suffer from the multiple comparisons problem, the necessity to specify the searchlight size as an additional parameter, and are—just like ROIs—insensitive to information encoded across distant brain regions.

Example call:

```
cfg.analysis = 'wholebrain';
% alternatives: 'ROI', 'searchlight'
```

TRAIN AND TEST CLASSIFIER

The core of the toolbox is training and testing the classifier. A classifier is typically first built from data where class membership is known (“train” classifier); then, the ability of the classifier to generalize to unseen data is evaluated (“test” or “validate” classifier). This separation is important to limit the impact of overfitting the classifier to the data and to have an unbiased estimate of the classifier’s generalization performance. The training and test cycle is hardcoded to make sure that training and test data are truly kept separate in each cycle (unless users create their own extension to overcome this separation, if required). TDT is equipped with a number of external classifiers which can be selected by setting the *cfg* variable. These classifiers belong to the packages LIBSVM (Chang and Lin, 2011) and LIBLINEAR (Fan et al., 2008) and include *L1*- and *L2*-norm regularized support vector machines and logistic regression. In addition, a correlation classifier (Haxby et al., 2001) has been implemented as part of the toolbox. Multiple classifiers have been compared across different data sets, with variable results (Cox and Savoy, 2003; Pereira et al., 2009; Misaki et al., 2010). Typically, the *L2*-norm support vector machine (SVM) performs quite well, which is why it is the default classifier in TDT. The options of the classifiers can be set using the *cfg* variable. TDT can also be used for purposes other than classification, for example to conduct a simple correlation of two data sets. Finally, at the stage of training and testing, advanced users can create an interface to completely external toolboxes that include other optimization methods (see How to Extend the Toolbox). All input can be passed as part of the settings in the *cfg* and all output can be stored for later processing.

Example call:

```
cfg.decoding.software = 'liblinear';
cfg.decoding.method = 'classification';
cfg.decoding.train.classification.
    model_parameters = '-s 0 -c 1 -q';
cfg.decoding.test.classification.
    model_parameters = '-q';
```

TRANSFORMATION OF RESULTS

Classifiers are essentially functions that describe a separation boundary (i.e., hyperplane, see **Table 1**) of multiple classes in voxel space. Newly predicted samples are mapped relative to this function and typically receive an output larger or smaller than 0 that denotes the distance to the separating function, where positive values denote membership to one class and negative values membership to the other class. For linear classifiers, so called decision values can be used to indicate the distance of a particular sample to the separating hyperplane. Using the predicted labels, the actual labels and the decision values, a number of metrics can be calculated to evaluate the performance of a classifier.

The most typical output for a decoding analysis is the mean cross-validated accuracy value. However, a number of other types of output are quite common and potentially useful. For example, it can be of interest to look at the classification accuracy for each class separately to assess classifier bias. A classification accuracy of 75% should be interpreted very differently when either class is decoded with 75% accuracy, or when one is decoded with 100% accuracy and the other with 50% (leading to 75% on average for a balanced number of samples in each class). In case of differently sized classes, balanced accuracy or *d*-prime can give indices of performance that take into account the different size of the groups. Additionally, the area under the receiver-operator-characteristic curve (AUC) which uses the distance of a classification output to the decision boundary can provide results about the information content using a graded rather than a binary response.

Other measures have received interest as well. For instance, weight maps are useful to provide information about the contribution of each voxel to the classification and are often used in whole-brain classification settings. Importantly, weight maps cannot be interpreted as reflecting the neural substrate relating to a task or another variable that is classified. Researchers interested in interpreting weights beyond the classifier alone can convert the weight maps to patterns which then show the contribution of each voxel for the representation of the classes under study (Haufe et al., 2014; Ritter et al., 2014). In addition, representational similarity analysis which exploits the correlation similarity structure of voxels and different distance metrics can be used to illustrate the representational distance of different conditions (Kriegeskorte et al., 2008; Nili et al., 2014). All approaches described above have been implemented in TDT.

Example call:

```
cfg.results.output = {'accuracy_minus_
    chance', 'AUC_minus_chance',
    'SVM_weights'};
cfg.decoding.method = 'classification';
% weights cannot be extracted when
% precomputed kernels are used, i.e.,
% we need to use the slower method of
% classification without kernels
```

STATISTICAL ANALYSIS

For a neuroscientist, it is important to statistically evaluate the results generated by a classification analysis: Is the observed result statistically significantly different from chance? Although at a first glance a decoding accuracy close to chance may seem disappointing, it is indeed quite commonly observed and not necessarily a problem. In neuroscience, typically the goal is not to maximize classification performance, but to demonstrate that information is present in the brain signals used for classification. This contrasts with the use in machine learning where the primary goal is to maximize classification performance and accuracies only slightly above chance would not be considered very useful. For that reason, there is not much work investigating the combination of multivariate decoding with classical statistics.

Importantly, it can be shown that combining results based on cross-validation and classical statistics can lead to overly liberal results. For example, cross-validated decoding accuracies are not

binomially distributed, which is why binomial tests should not be used to statistically evaluate cross-validation results (Schreiber and Krekelberg, 2013; Noirhomme et al., 2014). As an alternative, permutation testing can be used. However, for searchlight analyses this can become very time consuming. In addition, caution is warranted when using permutation approaches to make sure that these tests are not overly liberal (Stelzer et al., 2013), for example because they break up statistical dependencies at the wrong level (i.e., they violate the assumption of exchangeability, Nichols and Holmes, 2002).

These problems relate only to statistical analyses at the “decoding-level.” An alternative and common approach is to combine multiple decoding results and conduct a “second-level” statistical analysis. This is the case when evaluating whether the mean accuracy across a group of subjects is different from chance. For these purposes, classical parametric tests such as the Student’s *T*-test can be used.

In TDT, at the decoding-level both binomial testing and permutation testing have been implemented. However, binomial testing should only be used when training and test data are not reused, i.e., when no cross-validation and no bootstrapping is performed. A permutation test is conducted in three steps: First, the same decoding design is set up using a permutation scheme where data from different chunks is kept separate. Second, the permutations are conducted which can be run in parallel to speed-up processing. Third, the permutation test is conducted. Results can be reported in Matlab or written to brain images. For the second-level, users are advised to use dedicated third-party software such as SPSS, R, or Matlab which is specialized for statistical analyses and allows testing for basic assumptions of the tests. For users who do not want to export their results to other packages and are confident that the assumptions of their tests are met, TDT offers a set of basic functions that can conduct classical *T*-tests and *F*-tests.

Example call:

```
[results,cfg] = decoding(cfg);
    % run previously prepared decoding
cfg.design = make_design_permutation
    (cfg,1000,1); % creates one design
    with 1000 permutations
[reference,cfg] = decoding(cfg);
    % run permutations
cfg.stats.test = 'permutation'; % set test
cfg.stats.tail = 'right';
    % set tail of statistical correction
cfg.stats.output = 'accuracy_minus_chance';
    % choose from all original outputs
p = decoding_statistics
    (cfg,results,reference);
```

A DESCRIPTION OF OPTIONAL WORKFLOW STEPS—FOR ADVANCED LEVEL USERS

This section describes all remaining classification-related methods that can be carried out using a decoding analysis in TDT (Figure 1C). These include feature scaling, selection of model parameters in nested cross-validation, transformations of feature space, and feature selection. None of these steps are strictly necessary for a decoding analysis, but they can in principle help improving the results of a decoding analysis.

SCALING

Scaling is the process of adjusting the range of data which enters the classifier. This can be done to bring data to a range which improves the computational efficiency of the classifier (for example LIBSVM recommends scaling all data to be between 0 and 1). It can, however, also be used to change the relative contribution of individual features or individual samples or to remove the influence of the mean spatial pattern (Misaki et al., 2010; but see Garrido et al., 2013) which might affect classification performance. Scaling is also known as normalization, but we prefer the term scaling to distinguish it from another meaning of the term “normalization” which is commonly used in the MRI community to refer to spatial warping of images.

Typically, row scaling is used, i.e., scaling across samples within a given feature. Although scaling can theoretically improve decoding performance, for some data sets it may not have any influence (Misaki et al., 2010). Practically, scaling often has little or no influence on decoding performance when beta images or *z*-transformed data are passed, because this data already represents a scaled form of the raw images which is scaled relative to each run, rather than to all training data. However, scaling may still speed-up classification.

TDT allows a number of different settings: Either all data are scaled in advance (in TDT: “all”), which is only valid when scaling carries no information about class membership that influences test data, or scaling is carried out on training data only and these estimated scaling parameters are then applied to the test data (in TDT: “across”). The typically used scaling methods which have also been implemented in TDT are min0-max1 scaling or *z*-transformation. Min-max scaling scales all data to a range of 0 and 1, while *z*-transformation transforms data by removing the mean and dividing by the standard deviation. In addition to scaling data to a specified range, cut-off values can be provided for outlier reduction (Seymour et al., 2009). With this setting, all values larger than the upper cut-off are reduced to this limit, and all values smaller than the lower cut-off are set to this value. In TDT, these approaches can be combined with outlier reduction.

Example call:

```
cfg.scale.method = 'across';
    % scaling estimated on training data and
    applied to test data
cfg.scale.estimation = 'z';
    % z-transformation as scaling approach
cfg.scale.cutoff = [-3 3]; % all values
    > 3 are set = 3 (here: 3 standard
    deviations, because data is
    z-transformed)
```

PARAMETER SELECTION

Most multivariate classification and regression approaches use algorithms that contain parameters that need to be set by the user. For example, linear support vector machines have the regularization parameter *C* that determines the degree to which data points can cross the so-called *margin* (Table 1) which can even lead to misclassification during training (Cortes and Vapnik, 1995). Large values of *C* strongly penalize misclassification in training while small values of *C* allow for larger margins. This variable influences the bias-variance tradeoff, i.e., the tradeoff between

fitting a classifier too close to the training data (overfitting), or fitting it too little (underfitting, i.e., “ignoring” the structure of the training data too much), which directly affects the ability to generalize to the test data (Müller et al., 2001). Another example is the parameter *gamma* that needs to be set for non-linear radial basis function (RBF) SVMs which determines the width of the RBF kernel (Schölkopf et al., 1995). Since the optimal values for *C* and *gamma* are not known in advance, they are often estimated on the training data using *nested cross-validation*, a method where a cross-validation scheme is used only within training data to find good parameters. Typically, all parameter combinations across a range of target values are tried, a process known as *grid search*. The parameter combination that leads to best classification performance within training data is then applied to all training data and validated on the left-out test data. This has the consequence that for each real cross-validation fold a different set of optimal parameters might be chosen.

To avoid nested cross-validation, many researchers simply choose to set *C* to a fixed value of 1. In the typical scenario of an SVM where the number of features (e.g., voxels) is much larger than the number of samples (e.g., trials or beta images), in our experience only *C*-values much smaller than 1 have any influence on classification. This is probably because all samples within training data can easily be linearly separated, and for larger values of *C* the classifier yields stable solutions. Another problem is that with a small number of samples—even with nested cross-validation—the estimated parameters become quite variable, because they fit to the specifics of the training data and might not generalize well to test data. In particular cases, this might improve decoding performance slightly. Importantly, researchers should bear in mind that they must not escape the idea of cross-validation by trying out multiple parameter combinations and picking the one that yields the “best” results in the final cross-validated classification accuracy maps. Rather, they should use nested cross-validation for picking the optimal parameter and testing whether this choice generalizes to independent test data in the outer cross-validation loop. Alternatively, they could use parameters based on (previous) independent data sets or independent contrasts. Finally, the disadvantage of parameter selection is that it is quite time consuming, in particular when multiple parameters are selected concurrently. Parameter selection might make more sense in other circumstances or when other parameters are selected, for example the degree of a polynomial kernel.

In TDT, parameter selection is currently implemented as grid search, where all parameter combinations are combined. Depending on the problem, there might be smarter optimization approaches, but in our experience grid search works well for optimizing few parameters, in particular when only few samples are available. More advanced users can also create their own functions to tailor parameter selection to their specific problem.

Example call:

```
cfg.parameter_selection.method = 'grid';
% grid search (currently the only
  implemented method)
cfg.parameter_selection.parameters
= {'-c', '-g'};
```

```
cfg.parameter_selection.parameter_range =
{[0.0001 0.001 0.01 0.1 1 10 100 1000];
 [0.5 1 2]};
% The following parameters are set as
  defaults, i.e., they don't need to be
  called explicitly
cfg.parameter_selection.format.name =
'string_number';
cfg.parameter_selection.format.
  separator = ' ';
cfg.parameter_selection.optimization_
  criterion = 'max'; % pick value with
  maximal accuracy
```

FEATURE TRANSFORMATION

The term *feature transformation* is not commonly used in the brain decoding community, but it is known in the machine learning community as a set of methods that change the meaning of each individual feature (Liu and Motoda, 1998). We use this term to refer to a collection of methods that is used to change the space of data and possibly reduce the dimensionality of this changed space, as well. Principal component analysis (PCA) is one example. We prefer to treat such methods separately from *feature selection* (see below), because they lead to a change in the mapping of features. Methods that we refer to as *feature selection* (below) select a subset of features (e.g., voxels) of the input. In contrast, in *feature transformation* a data point refers to a non-linear transformation applied to the input, which essentially creates new features. For example, the outcome of a principle component analysis (PCA) is no longer separate single voxel data, but a linear combination of the original voxel data. This strict semantic separation helps to avoid confusion, makes the results of feature selection easier to interpret, facilitates the addition of new methods, and facilitates the use of multi-step feature selection (e.g., first running PCA and transforming voxels to principle component space, and then running feature selection on a subset of principal components). Other examples of feature transformation would encompass independent component analysis (ICA), Fourier transformation, or pre-calculated mappings that can for example be used to align data spaces between subjects, known as hyperalignment (Haxby et al., 2011). Currently, only PCA is implemented in TDT, but other methods can easily be added. The transformation can be estimated on training data only and applied to test data in each cross-validation step (“across”). Alternatively, it can be estimated on both training and test data (“all”) when it is ensured that this does not lead to non-independence (Kriegeskorte et al., 2009). This should not be the case as long as no label information is used for transformations (which is not done in PCA). If feature transformation is also used for dimensionality reduction, the user can specify either the number of to be selected features or can specify a critical value in the score of the method that needs to be exceeded for a transformed feature to be of interest (e.g., for PCA the percent variance explained).

Example call:

```
cfg.feature_transformation.method = 'PCA';
cfg.feature_transformation.
  estimation = 'all';
cfg.feature_transformation.critical_value =
```

```
0.1; % only keep components that
explain at least 10 percent variance
```

FEATURE SELECTION

Books have been written about *feature selection* (Guyon et al., 2006) which is a method in machine learning that refers to the reduction of the dimensionality of data, with the aim of finding the most relevant features and for improving classification performance. Feature selection has been classified into three general categories: filters, wrappers, and embedded methods (Guyon and Elisseeff, 2003). Filters are methods that use univariate or multivariate statistics based on the data to rank features and select them based on their rank. Examples are the F-statistic, the weights of a classifier, or in brain imaging external masks that provide a ranking, such as external functional localizer images. Wrappers are methods that iteratively include or exclude features, based on some optimization criterion specific to the feature selection method. Examples are sequential forward selection or sequential backward elimination. Finally, embedded methods are methods where the selection of features becomes part of the classification problem. Examples are LASSO (Tibshirani, 1996) or L1-regularized support vector machines (Fan et al., 2008) where unimportant voxels receive a weight of zero, which eliminates their contribution. Recursive feature elimination is another popular method (De Martino et al., 2008) and is sometimes referred to as a wrapper method, although the final feature set and classification can depend on the previous steps. For that reason—and to better distinguish it from other sequential methods—it can also be referred to as an embedded method.

In TDT, feature selection has been implemented with a number of filter methods and with the embedded method of recursive feature elimination. Implemented filter methods are the F-statistic, the Mann-Whitney U-statistic, classifier weights, and external masks, with the additional option of supplying a separate mask for each cross-validation fold. For determining the optimal feature set, nested cross-validation can be performed. It should be noted, however, that similar to parameter selection the utility of feature selection is limited when the number of samples is very small. In that case, nested cross-validation can optimize the number of features to the idiosyncrasies of the data, and it might be better not to perform feature selection at all. In addition, feature selection can be computationally highly expensive and can thus dramatically slow down classification, in particular when recursive feature elimination is used with nested cross-validation and a large number of steps. As has been mentioned for parameter selection, researchers should be cautious not to try out a number of different feature selection methods and choose one that produces the “best” final result. Often researchers might be interested in implementing multiple feature selection steps. In the current version of TDT, up to two feature selection steps are possible. More sophisticated feature selection methods can be added to TDT, e.g., by including them directly into custom classification routines.

Example call:

```
cfg.feature_selection.method = 'filter';
cfg.feature_selection.filter = 'F';
```

```
cfg.feature_selection.n_vox = 'automatic';
% nested CV to determine optimal number
of features
```

More sophisticated example call:

```
% First, run feature selection using
external ranking mask
cfg.feature_selection.feature_selection.
method = 'filter';
cfg.feature_selection.feature_selection.
filter = 'external';
cfg.feature_selection.feature_selection.
external_fname = 'mylocalizer.img';
cfg.feature_selection.feature_selection.
n_vox = 0.5; % select 50% of all voxels
% Then run Recursive Feature Elimination on
the remaining voxels
cfg.feature_selection.method = 'embedded';
cfg.feature_selection.embedded = 'RFE';
% Recursive Feature Elimination
cfg.feature_selection.n_vox =
[5 10 20 40 80 160]; % possible target
values for number of voxels in RFE
(i.e., where RFE may stop in the end)
cfg.feature_selection.nested_n_vox =
[5:5:200]; % eliminate 5 voxels per
step in nested CV
```

HOW TO EXTEND THE TOOLBOX

TDT comes with a number of processing approaches, but of course not all methods have been implemented. Researchers might want to use their preferred machine learning algorithm or feature selection method for their decoding analysis. Other users might wish to use TDT only as a wrapper tool for their own machine learning package. Data can also be passed to TDT, which in principle extends TDT to other modalities such as EEG or MEG decoding. In this section, we will use three examples to illustrate how TDT can be extended.

EXAMPLE 1: INTRODUCE A NEW CLASSIFIER

To add new classifiers, two functions need to be provided, one for training and one for testing the classifier. If the new classifier should be called “newclassifier,” the training method needs to be saved as `newclassifier_train.m` and is called by

```
model = newclassifier_train
(labels_train,data_train, cfg)
```

where *model* is any information that is needed to evaluate the classifier. Within the function, the *cfg* is needed to distinguish whether a classification or regression should be performed, or whether a precomputed kernel is passed. In addition, the parameters for classification can be passed in this variable. In the following, we show pseudocode for the function to implement a new classifier.

```
function model = newclassifier_train
(labels_train,data_train, cfg)
switch lower(cfg.decoding.method)
case 'classification'
model = your_train_algorithm(labels_train,
data_train, cfg.decoding.train,
classification.model_parameters)
```

```

case 'classification_kernel'
    ...
case 'regression'
    ...
otherwise
    error('...')
end

```

If the external function creates several outputs, these can be grouped within the variable *model*. The structure for the second function parallels the first and looks as follows:

```

function decoding_out = newclassifier_test
    (labels_test, data_test, cfg, model)
switch lower(cfg.decoding.method)
case 'classification'
[predicted_labels decision_values
 other_output] = your_test_algorithm
    (labels_test, data_test, model,
    cfg.decoding.test.classification.
    model_parameters);
decoding_out.predicted_labels =
    predicted_labels;
decoding_out.true_labels = labels_test;
decoding_out.decision_values =
    decision_values;
decoding_out.model = model;
decoding_out.opt = other_output;
    % optional other output which we added
    just as an example
(...)
end

```

It is important that *decoding_out* returns a struct variable (i.e., structure array) with the subfields above (predicted_labels, true_labels, decision_values, model, opt). If some of these are not provided by a classifier, empty arrays can be passed. These new classifier functions can then be called in any decoding analysis by setting

```
cfg.decoding.software = 'newclassifier';
```

EXAMPLE 2: INTRODUCE A NEW RESULT MEASURE

New, more sophisticated means of transforming results can also be introduced to TDT. For example, consider a user who wants to weight classification accuracies by decision values before averaging. The function would be written as follows:

```

function output = transres_dvtimesacc
    (decoding_out, chancelevel, cfg, data)
predicted_labels = vertcat
    (decoding_out.predicted_labels);
true_labels = vertcat
    (decoding_out.true_labels);
accuracy = predicted_labels==true_labels;
decision_values = vertcat
    (decoding_out.decision_values);
dv_norm = abs(decision_values)/
    max(abs(decision_values));
output = 100 * accuracy.* dv_norm;

```

The function should be saved as *transres_dvtimesacc.m* and can then be used by setting

```
cfg.results.output = 'dvtimesacc';
```

EXAMPLE 3: EXTEND TOOLBOX TO EXTERNAL MACHINE LEARNING PACKAGE OR MORE COMPLEX PROCESSING STREAMS

The toolbox can be interfaced with complete external machine learning packages, for example, the powerful Shogun toolbox (Sonnenburg et al., 2010). This might be useful for users who want to use the general workflow of TDT, e.g., the data handling, cross-validation facilities and searchlight routines, but do not wish to implement a large number of external algorithms for use in TDT. The same general procedure can be applied to use highly sophisticated processing streams that do not directly fit into the TDT framework. There are two approaches for creating an interface to an external toolbox. The first takes place at the level of the classifier. Essentially, an interface is created where all steps of the decoding analysis—if requested, even including cross-validation—are carried out outside of TDT. The training function of a classifier acts as a placeholder to pass data to the testing function.

```

function model = external_toolbox_train
    (labels_train, data_train, cfg)
model.data_train = data_train;
model.labels_train = labels_train;

```

The interface function would look as follows:

```

function decoding_out =
    external_toolbox_test
    (labels_test, data_test, cfg, model)
[predicted_labels accuracy decision_values]
= externalpackagewrapper(data_train,
    labels_train, data_test,
    labels_test, cfg)

```

The function *externalpackagewrapper* would then contain the script which is normally executed using the external software package. This can be made general purpose by passing parameters using the setting *cfg.decoding.train.classification.model_parameters* of TDT.

The second approach for creating an interface to external toolboxes takes place at the level of the transformation of outputs. This might be useful if a user would like to complete the entire decoding analysis including the generation of output in an external toolbox. In this case, the user creates a placeholder for the classifier which merely passes all data and classification parameters inside the *model* variable as the optional output *decoding_out.opt* (e.g., using the toolbox function “passdata” as a classifier). Then, the interface to the external package is created as follows:

```

function output = transres_packageinterface
    (decoding_out, chancelevel, cfg, data)
output = externalpackagewrapper
    (decoding_out.opt, data)

```

where *output* contains all desired output of the external package.

TOOLBOX VALIDATION: SIMULATIONS AND APPLICATIONS TO fMRI DATA

To demonstrate that TDT works correctly, we validated the toolbox by running a number of analyses on simulated and real data. These analyses are not supposed to provide an overview over all

capabilities of the toolbox. Rather, they are used to illustrate some of the functionalities of TDT.

SIMULATIONS: SEARCHLIGHT ANALYSIS, WHOLE-BRAIN ANALYSIS, REGION-OF-INTEREST ANALYSIS

Simulated data consisted of 48 ellipsoid volumes (31,016 voxels), centered in volumes of $64 \times 64 \times 16$ voxels (Figure 4). Each voxel was assigned independent Gaussian random noise (mean = 0, standard deviation = 1). Half of the volumes were assigned to the “noise” category and remained unchanged. The other half was assigned to the “signal” category where a pattern was added to axial slice 8. This pattern consisted of Gaussian random noise (mean = 0, standard deviation = 0.8). The same pattern was added to all signal volumes. Data was then split in 8 runs of 6 volumes each, providing three samples per category in each run. Please note that noise is spatially and temporally independent, which is a strong simplification but is sufficient for the purpose of this simulation. Additionally, univariate effects are present which is not necessary for classification to work. The difference of the mean results is shown in Figure 4A (left panel).

For Simulation 1, we ran a searchlight analysis with a leave-one-run-out cross-validation scheme. The searchlight had a radius of 2 voxels, encompassing 27 voxels. The classifier was a linear L2-norm SVM with $C = 1$, in the implementation of LIBSVM (Chang and Lin, 2011). As output, the mean classification accuracy was returned. The results are shown in Figure 4A (right panel). As expected, the accuracies around the signal regions are highly above chance while in other regions they fluctuate around chance-level. In addition, Figure 4A (right panel) illustrates the slight smoothing imposed by the searchlight, where the high SNR spreads the word “TDT” to the neighboring axial slices (Etzel et al., 2013).

Simulation 2 consisted of two whole-brain analyses that were run on the same simulated data set used for Simulation 1. In Simulation 2A, we used all data to generate a weight map, i.e., we did not separate data in training and test sets. A weight map indicates the contribution of each voxel to the classification which reflects a combination of signal enhancement and noise suppression. As can be seen in Figure 4B (left panel), the results are similar to the difference in means of the original data, because noise was spatially uncorrelated. Simulation 2B used the same leave-one-run-out cross-validation scheme as in the searchlight analysis to achieve one classification accuracy for the entire brain. The analysis consisted of an additional nested cross-validation where we conducted recursive feature elimination to identify the set of voxels that is optimally suited for carrying out the classification task. Classification performance was at ceiling (100 % accuracy). Figure 4B (right panel) shows how often each voxel is chosen in any of the six cross-validation steps for recursive feature elimination, again clearly favoring signal voxels over noise voxels.

In Simulation 3, we ran a region-of-interest (ROI) analysis with two ROIs, where one ROI covered one third of the signal region while the other covered the remaining two thirds (Figure 4C, left panel). Again, we used a leave-one-run out cross-validation scheme. We then continuously varied the amount of signal that was added to the noise. As can be seen in Figure 4C (right panel), the decoding accuracy gradually decreased with

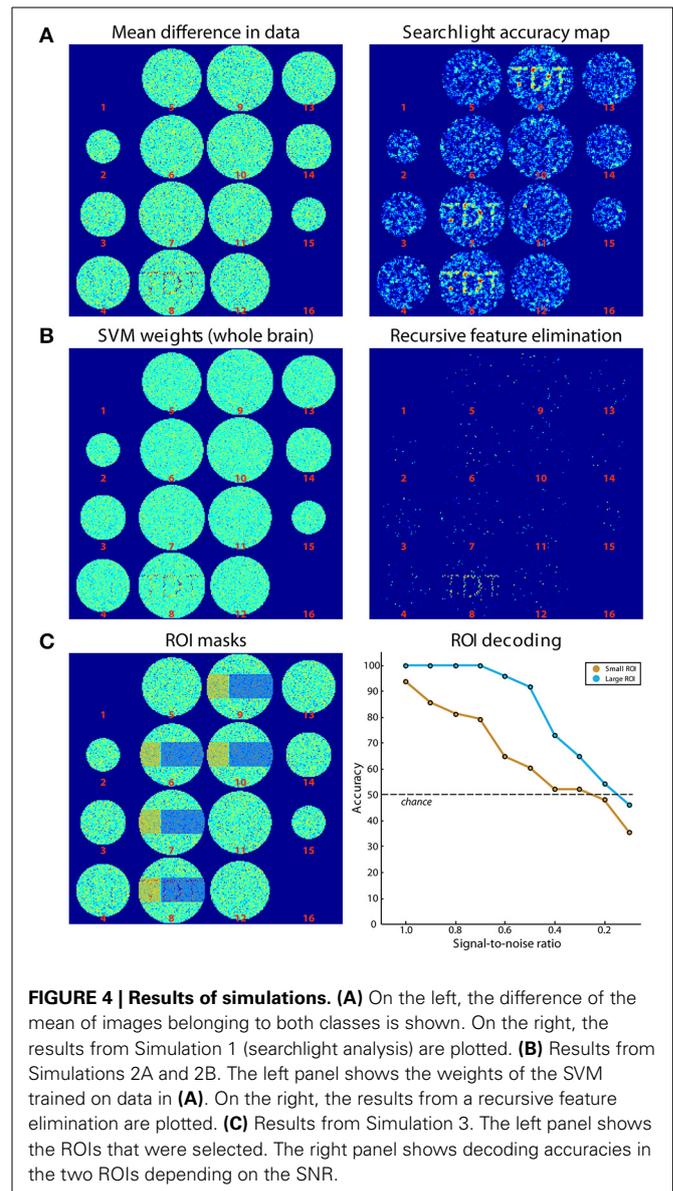


FIGURE 4 | Results of simulations. (A) On the left, the difference of the mean of images belonging to both classes is shown. On the right, the results from Simulation 1 (searchlight analysis) are plotted. **(B)** Results from Simulations 2A and 2B. The left panel shows the weights of the SVM trained on data in (A). On the right, the results from a recursive feature elimination are plotted. **(C)** Results from Simulation 3. The left panel shows the ROIs that were selected. The right panel shows decoding accuracies in the two ROIs depending on the SNR.

decreasing SNR until it reached chance-level when no signal is present. In addition, the accuracy in the smaller ROI was generally lower than that in the larger ROI, reflecting the reduced amount of signal present in that region.

VALIDATION ON EMPIRICAL DATA: THE Haxby 2001 DATASET

To validate the toolbox on real imaging data, we used data from a study by Haxby et al. (2001) which is publicly available (<http://data.pymvpa.org/datasets/haxby2001/>). Because of space limitations, we do not use the dataset provided with the toolbox for this article, because it would require more detailed explanation of the experimental paradigm. Haxby et al. (2001) investigated processing of images belonging to 8 different object categories in ventral visual cortex. Data from six subjects were made available. For a complete description of the experimental paradigm and imaging parameters, see the original publication.

The experiment consisted of 12 runs where each category was shown once in each run in a 24 s block. Data were motion corrected and detrended using SPM12 (<http://www.fil.ion.ucl.ac.uk/spm/software/spm12/>). We fitted each block with a canonical hemodynamic response function, yielding one beta image per condition per run and a total of 96 beta images per subject.

For the first empirical analysis, we ran a ROI decoding in ventral temporal cortex. Our goal was to investigate the classification accuracy and the confusion of the different categories. For that purpose, we used the masks provided with the Haxby data set to focus our analysis on ventral temporal cortex. The beta images were then submitted to a leave-one-run-out cross-validation scheme, using a one-vs.-one multiclass SVM classification approach². The output was specified to reveal a confusion matrix comparing the frequency of the predicted label with the true label. This approach was repeated for all six subjects and the resulting confusion matrices were averaged. The result is shown in **Figure 5A**. As can be seen, each class could be classified very well, with only little confusion between the classes. The results indicate two clusters with larger confusion: Images of faces and cats were confused more often than other categories, and images of inanimate objects were confused more often. This could indicate that in ventral visual cortex, faces and cats as well as different inanimate objects are processed more similarly to each other.

For the second empirical analysis, we ran a multiclass searchlight analysis on subject 1 from the data set, with a leave-one-run out cross-validation scheme and a searchlight radius of 4 voxels.

To assess statistical significance, we ran a permutation test, yielding a critical cut-off of 25 % accuracy (chance-level: 12.5 %) and a cluster size of $k = 24$ voxels (cluster-level corrected $p < 0.001$). The resulting accuracy maps were masked by the cut-off and are shown in **Figure 5B**. Large portions of the dorsal and ventral visual cortex carry information about the different categories, with decoding accuracies reaching 62.5 % ventral temporal cortex of subject 1. Taken together, these results fit nicely with the known architecture of the visual cortex and reflect the feasibility of the toolbox to reveal information about the content of representations in the human brain.

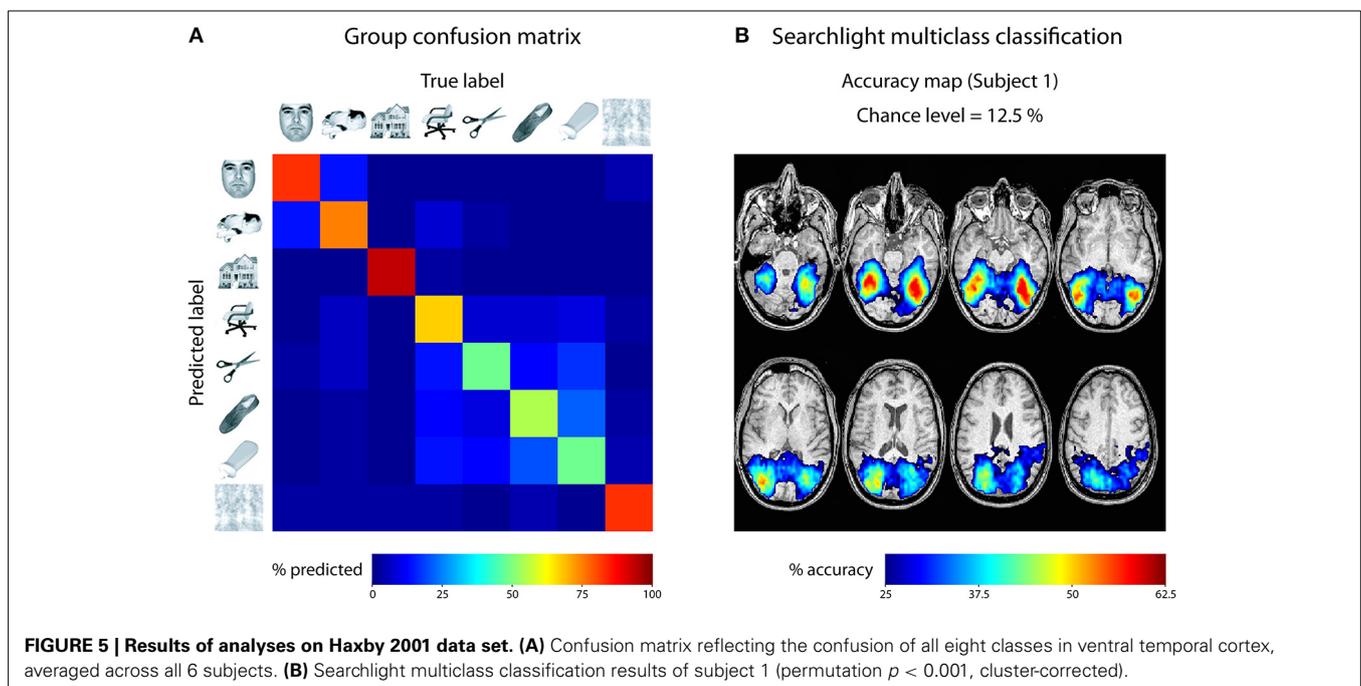
DISCUSSION

We hope that the above illustration of TDT demonstrated the simplicity and general utility of this toolbox for multivariate analysis of functional MRI data. In addition, the extensive error checking helps prevent many programming errors and should guide users in how to quickly resolve them. We did not go into detail about an additional important feature of TDT that proved helpful to prevent errors: the visualization of the decoding design. It shows users at a glance if the cross-validation design they intended to set-up is indeed the one that is computed and if the input data is indeed the data they wanted to use. This further facilitates the prevention of unwanted or erroneous analyses. To help in getting started, TDT comes with an example dataset and example analysis scripts.

FEATURES NOT CONTAINED IN TDT: PREPROCESSING AND RESULTS VISUALIZATION

When we developed TDT, we initially used it as a tool for conducting searchlight analyses on preprocessed data in order to create individual searchlight accuracy maps that could then be used as input to group-level analyses. Later, the toolbox was extended

²Please note that one-vs.-one multiclass SVM classification usually follows a “winner-takes-all” voting scheme. In case of ties, LIBSVM automatically assigns the first class to the tied output. To circumvent this bias, TDT resolves ties by choosing the class with the maximum decision value over all pairs of comparisons.



to become more general purpose, adding whole-brain and ROI analyses and more and more machine-learning related utilities. Our focus for TDT was on specifically creating a tool that provides users with the means to conduct multivariate decoding. Most decoding analyses are carried out on preprocessed data (e.g., spatially realigned or temporally detrended data), but preprocessing is not a part of TDT, as numerous software packages exist that have been created for that purpose, each with their own benefits and drawbacks. For example, detrending data or scaling time-series are important steps for single image decoding, but this can be done with popular software packages including SPM, FSL, or AFNI or directly in Matlab with specialized toolboxes. The same applies to visualization of results: third-party software can be used for that purpose, for example MRICron (Rorden, 2007) to visualize searchlight accuracy maps or weight maps.

COMPARISON TO EXISTING PACKAGES OF MULTIVARIATE DATA ANALYSIS

A few other toolboxes have been created that can be used to carry out multivariate decoding analyses on fMRI data. The key advantages of TDT have been listed in the introduction. In general, it is difficult to tell to which degree other toolboxes offer similar advantages, for example where they might be comparably transparent, fast, easy to use, or versatile. More objective measures that have been used for comparison purposes have partly underestimated the ability of other toolboxes (Schrouff et al., 2013; Grotegerd et al., 2014) because the toolboxes might be more elaborate than described on the toolbox websites. Rather than thoroughly describing the advantages and disadvantages of all existing toolboxes, we will mention the most widely used and in our view most promising alternative toolboxes and try to elucidate some degree of comparison to TDT.

The Princeton MVPA toolbox (<http://code.google.com/p/princeton-mvpa-toolbox/>) is a rather versatile tool with a number of classifiers, basic feature selection capabilities and scaling and the possibility to run searchlight analyses, but it requires a more advanced level of programming skills and to our knowledge is not further developed. Another advantage of this toolbox is an active user community. TDT on the other hand offers more analysis methods, can be easily extended and due to the multilevel approach is probably easier for getting started. The SPM interface of TDT is particularly well suited for users of SPM.

The developers of the Princeton MVPA toolbox seem to have shifted their focus to PyMVPA (Hanke et al., 2009a,b), a highly versatile programming environment that allows for a large variety of decoding analyses. The key advantages of this toolbox are the high level of sophistication, the active user community and the fact that the toolbox does not require third-party software. While TDT requires the third-party software Matlab and might not be as versatile in its core functions, PyMVPA also requires more advanced programming skills and for searchlight analyses is much slower than TDT. In addition, Matlab is still widespread in the neuroimaging community, and running decoding analyses on SPM results from Python is not straightforward. Finally, TDT can also be easily extended if users need additional functionality, which also requires only little programming skills (see Example 3:

Extend Toolbox to External Machine Learning Package or More Complex Processing Streams).

A more recent development is PRoNTo (Pattern Recognition of Neuroimaging Toolbox, Schrouff et al., 2013) which has the benefit of a graphical user interface and which also comes with an SPM interface. However, the structure of the toolbox is not optimized for searchlight analyses, and parameter selection or feature selection have not been implemented. Although TDT does not offer a graphical user interface, we are currently working on an interface through the SPM batch system which should become available with the next release of the toolbox and which obviates any command line programming knowledge.

We would also like to mention a toolbox specifically created for carrying out representational similarity analysis (RSA toolbox, Nili et al., 2014) which can serve as a standard for this type of analysis. The capabilities of TDT for representational similarity analysis at the current stage are still much more basic than those of the RSA toolbox. However, TDT is in our experience faster than the RSA toolbox, both for searchlight analyses and for creating correlation matrices. Users who would like to benefit from the speed of TDT, but use all functionalities of the RSA toolbox might consider using both toolboxes. Alternatively, appropriate extensions can be made (see How to Extend the Toolbox) to include many functionalities of the RSA toolbox. We are planning to include more representational similarity analysis methods in future versions of TDT.

A very recent development under Matlab is CosmoMVPA (<http://cosmomvpa.org/>) which is still in its very early stage. The general structure of the toolbox is quite similar to TDT. Rather than creating a *cfg* in the beginning as is the case for TDT, each option is passed along with the multiple subfunctions that are called. The toolbox offers a range of classifiers, allows volumetric and surface-based searchlight analysis and has a basic interface to SPM and other brain analysis software packages. The toolbox is easy to use with some intermediate level programming knowledge in Matlab. However, CosmoMVPA currently does not offer decoding designs, potentially leading to overlooked mistakes in decoding analyses. Due to its novelty, the error management of the toolbox and additional methods such as feature selection and parameter selection are still quite basic. Passing parameters along the use of subfunctions also invites mistakes on the side of the user. However, it is possible that these current limitations will be changed in future versions of the toolbox.

Finally, for completeness we would like to mention additional MVPA software packages for fMRI data: Searchmight (<http://www.princeton.edu/~fpereira/searchmight/>) which is a dedicated and fast software package for simple searchlight analyses with different classifiers³, 3dsvm (http://afni.nimh.nih.gov/pub/dist/doc/program_help/3dsvm.html) which is part

³It is difficult to compare the Searchmight and TDT in terms of speed. Since the major bottleneck in processing speed is the classifier, this will depend crucially on the implementation. For example, Searchmight comes with very fast and dedicated Gaussian Naïve Bayes searchlight capabilities, but at the same time does not pass kernel matrices to LIBSVM which can slow down processing.

of the software package AFNI; PROBID (<http://www.kcl.ac.uk/iop/depts/neuroimaging/research/imaginganalysis/Software/PROBID.aspx>) which is specialized for group comparisons, MANIA (Grotegerd et al., 2014), and the brain decoder toolbox (<http://www.cns.atr.jp/dni/download/brain-decoder-toolbox/>).

CONCLUSIONS

TDT offers a user-friendly, yet powerful and flexible framework for the multivariate analysis of brain imaging data. The toolbox has many advantages in terms of structure, transparency, speed and error management. It comes with an interface to the common brain data analysis software SPM which should make it particularly easy to apply to existing data. Beginners can start using the toolbox with one single line of code, which increasingly can be extended to exploit the full functionality of TDT. In addition, the toolbox can easily be extended for more general purpose use which allows adding new classifiers, feature selection methods, or even complete external software packages for multivariate data analysis. We hope that TDT—through its simplicity and flexibility—will encourage a much broader application of machine learning methods in the analysis of functional imaging data.

LICENSE STATEMENT

TDT can be downloaded from <http://www.bccn-berlin.de/tdt>. The toolbox code is open source, but is licensed as copyright software under the terms of the GNU General Public License (Free Software Foundation). In addition, the toolbox comes with third-party software (LIBSVM, LIBLINEAR, Newton-SVM, <http://research.cs.wisc.edu/dmi/svm/nsvm/>), each with their respective copyright. TDT has been tested under Windows, Linux, and OS X and works both on 32 and 64 bit systems. The toolbox is compatible with Matlab versions R2006b or above and requires no additional Matlab toolboxes. It works “out-of-the-box” with an installed version of SPM2, SPM5, SPM8, or SPM12b. In addition, we provide an example data set that can be downloaded from the toolbox website.

ACKNOWLEDGMENTS

The authors declare no conflict of interest. We thank Carsten Bogler, Thomas Christophel, Stephan Geuter, Elliott Wimmer and both reviewers for helpful comments on earlier versions of the manuscript. We thank the members of the Haynes lab and many volunteers from around the world for trying pre-publication releases of the toolbox. This work was supported by the “Bernstein Award for Computational Neuroscience” by the German Ministry of Education and Research (BMBF Grant No. 01GQ1006) and by the German Research Foundation (DFG Grant GRK1589/1).

REFERENCES

- Allefeld, C., and Haynes, J.-D. (2014). Searchlight-based multi-voxel pattern analysis of fMRI by cross-validated MANOVA. *Neuroimage* 89, 345–357. doi: 10.1016/j.neuroimage.2013.11.043
- Averbeck, B. B., Latham, P. E., and Pouget, A. (2006). Neural correlations, population coding and computation. *Nat. Rev. Neurosci.* 7, 358–366. doi: 10.1038/nrn1888
- Bilalić, M., Grottenhaler, T., Nägele, T., and Lindig, T. (in press). The faces in radiological images: fusiform face area supports radiological expertise. *Cereb. Cortex*. doi: 10.1093/cercor/bhu272
- Blankertz, B., Dornhege, G., Krauledat, M., Müller, K.-R., and Curio, G. (2007). The non-invasive Berlin brain–computer interface: fast acquisition of effective performance in untrained subjects. *Neuroimage* 37, 539–550. doi: 10.1016/j.neuroimage.2007.01.051
- Chang, C. C., and Lin, C. J. (2011). LIBSVM: a library for support vector machines. *ACM Trans. Intell. Syst. Technol.* 2, 1–27. doi: 10.1145/1961189.1961199
- Christophel, T. B., Cichy, R. M., Hebart, M. N., and Haynes, J.-D. (in press). Parietal and early visual cortices encode working memory content across mental transformations. *Neuroimage*. doi: 10.1016/j.neuroimage.2014.11.018
- Christophel, T. B., and Haynes, J.-D. (2014). Decoding complex flow-field patterns in visual working memory. *Neuroimage* 91, 43–51. doi: 10.1016/j.neuroimage.2014.01.025
- Christophel, T. B., Hebart, M. N., and Haynes, J.-D. (2012). Decoding the contents of visual short-term memory from human visual and parietal cortex. *J. Neurosci.* 32, 12983–12989. doi: 10.1523/JNEUROSCI.0184-12.2012
- Cortes, C., and Vapnik, V. (1995). Support-vector networks. *Mach. Learn.* 20, 273–297. doi: 10.1007/BF00994018
- Cox, D. D., and Savoy, R. L. (2003). Functional magnetic resonance imaging (fMRI) “brain reading”: detecting and classifying distributed patterns of fMRI activity in human visual cortex. *Neuroimage* 19, 261–270. doi: 10.1016/S1053-8119(03)00049-1
- Cui, Y., Liu, B., Luo, S., Zhen, X., Fan, M., Liu, T., et al. (2011). Identification of conversion from mild cognitive impairment to Alzheimer’s disease using multivariate predictors. *PLoS ONE* 6:e21896. doi: 10.1371/journal.pone.0021896
- De Martino, F., Valente, G., Staeren, N., Ashburner, J., Goebel, R., and Formisano, E. (2008). Combining multivariate voxel selection and support vector machines for mapping and classification of fMRI spatial patterns. *Neuroimage* 43, 44–58. doi: 10.1016/j.neuroimage.2008.06.037
- Etzel, J. A., Gazzola, V., and Keysers, C. (2009). An introduction to anatomical ROI-based fMRI classification analysis. *Brain Res.* 1282, 114–125. doi: 10.1016/j.brainres.2009.05.090
- Etzel, J. A., Zacks, J. M., and Braver, T. S. (2013). Searchlight analysis: promise, pitfalls, and potential. *Neuroimage* 78, 261–269. doi: 10.1016/j.neuroimage.2013.03.041
- Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: a library for large linear classification. *J. Mach. Learn. Res.* 9, 1871–1874. Available online at: <http://dl.acm.org/citation.cfm?id=1442794>
- Formisano, E., De Martino, F., and Valente, G. (2008). Multivariate analysis of fMRI time series: classification and regression of brain responses using machine learning. *Magn. Reson. Imaging* 26, 921–934. doi: 10.1016/j.mri.2008.01.052
- Friston, K. J., Holmes, A. P., Worsley, K. J., Poline, J.-P., Frith, C. D., and Frackowiak, R. S. (1994). Statistical parametric maps in functional imaging: a general linear approach. *Hum. Brain Mapp.* 2, 189–210. doi: 10.1002/hbm.460020402
- Friston, K. J., Rotshtein, P., Geng, J. J., Sterzer, P., and Henson, R. N. (2006). A critique of functional localisers. *Neuroimage* 30, 1077–1087. doi: 10.1016/j.neuroimage.2005.08.012
- Garrido, L., Vaziri-Pashkam, M., Nakayama, K., and Wilmer, J. (2013). The consequences of subtracting the mean pattern in fMRI multivariate correlation analyses. *Front. Neurosci.* 7:174. doi: 10.3389/fnins.2013.00174
- Grotegerd, D., Redlich, R., Almeida, J. R., Riemenschneider, M., Kugel, H., Arolt, V., et al. (2014). MANIA—A pattern classification toolbox for neuroimaging data. *Neuroinformatics* 12, 471–486. doi: 10.1007/s12021-014-9223-8
- Guggenmos, M., Thoma, V., Cichy, R. M., Haynes, J.-D., Sterzer, P., Richardson-Klavehn, A. (in press). Non-holistic coding of objects in lateral occipital complex with and without attention. *Neuroimage*. doi: 10.1016/j.neuroimage.2014.12.013
- Guyon, I., and Elisseeff, A. (2003). An introduction to variable and feature selection. *J. Mach. Learn. Res.* 3, 1157–1182. Available online at: <http://dl.acm.org/citation.cfm?id=944968>
- Guyon, I., Gunn, S., Nikravesh, M., and Zadeh, L. (2006). *Feature Extraction: Foundations and Applications*. Berlin: Springer.
- Hanke, M., Halchenko, Y. O., Sederberg, P. B., Hanson, S. J., Haxby, J. V., and Pollmann, S. (2009a). PyMVPA: a python toolbox for multivariate pattern analysis of fMRI data. *Neuroinformatics* 7, 37–53. doi: 10.1007/s12021-008-9041-y

- Hanke, M., Halchenko, Y. O., Sederberg, P. B., Olivetti, E., Fründ, I., Rieger, J. W., et al. (2009b). PyMVPA: a unifying approach to the analysis of neuroscientific data. *Front. Neuroinform.* 3:3. doi: 10.3389/neuro.11.003.2009
- Haufe, S., Meinecke, F., Görgen, K., Dähne, S., Haynes, J.-D., Blankertz, B., et al. (2014). On the interpretation of weight vectors of linear models in multivariate neuroimaging. *Neuroimage* 87, 96–110. doi: 10.1016/j.neuroimage.2013.10.067
- Haxby, J. V., Gobbini, M. I., Furey, M. L., Ishai, A., Schouten, J. L., and Pietrini, P. (2001). Distributed and overlapping representations of faces and objects in ventral temporal cortex. *Science* 293, 2425–2430. doi: 10.1126/science.1063736
- Haxby, J. V., Guntupalli, J. S., Connolly, A. C., Halchenko, Y. O., Conroy, B. R., Gobbini, M. I., et al. (2011). A common, high-dimensional model of the representational space in human ventral temporal cortex. *Neuron* 72, 404–416. doi: 10.1016/j.neuron.2011.08.026
- Haynes, J.-D., and Rees, G. (2006). Decoding mental states from brain activity in humans. *Nat. Rev. Neurosci.* 7, 523–534. doi: 10.1038/nrn1931
- Haynes, J.-D., Sakai, K., Rees, G., Gilbert, S., Frith, C., and Passingham, R. E. (2007). Reading hidden intentions in the human brain. *Curr. Biol.* 17, 323–328. doi: 10.1016/j.cub.2006.11.072
- Hebart, M. N., Donner, T. H., and Haynes, J.-D. (2012). Human visual and parietal cortex encode visual choices independent of motor plans. *Neuroimage* 63, 1393–1403. doi: 10.1016/j.neuroimage.2012.08.027
- Hebart, M. N., Schriever, Y., Donner, T. H., and Haynes, J.-D. (in press). The relationship between perceptual decision variables and confidence in the human brain. *Cereb. Cortex* bhu181. doi: 10.1093/cercor/bhu181
- Hesselmann, G., Hebart, M., and Malach, R. (2011). Differential BOLD activity associated with subjective and objective reports during “blindsight” in normal observers. *J. Neurosci.* 31, 12936–12944. doi: 10.1523/JNEUROSCI.1556-11.2011
- Horikawa, T., Tamaki, M., Miyawaki, Y., and Kamitani, Y. (2013). Neural decoding of visual imagery during sleep. *Science* 340, 639–642. doi: 10.1126/science.1234330
- Kanwisher, N., McDermott, J., and Chun, M. M. (1997). The fusiform face area: a module in human extrastriate cortex specialized for face perception. *J. Neurosci.* 17, 4302–4311.
- Kriegeskorte, N. (2011). Pattern-information analysis: from stimulus decoding to computational-model testing. *Neuroimage* 56, 411–421. doi: 10.1016/j.neuroimage.2011.01.061
- Kriegeskorte, N., Formisano, E., Sorger, B., and Goebel, R. (2007). Individual faces elicit distinct response patterns in human anterior temporal cortex. *Proc. Natl. Acad. Sci. U.S.A.* 104, 20600–20605. doi: 10.1073/pnas.0705654104
- Kriegeskorte, N., Goebel, R., and Bandettini, P. (2006). Information-based functional brain mapping. *Proc. Natl. Acad. Sci. U.S.A.* 103, 3863–3868. doi: 10.1073/pnas.0600244103
- Kriegeskorte, N., Mur, M., and Bandettini, P. (2008). Representational similarity analysis—Connecting the branches of systems neuroscience. *Front. Syst. Neurosci.* 2:4. doi: 10.3389/neuro.06.004.2008
- Kriegeskorte, N., Simmons, W. K., Bellgowan, P. S., and Baker, C. I. (2009). Circular analysis in systems neuroscience: the dangers of double dipping. *Nat. Neurosci.* 12, 535–540. doi: 10.1038/nn.2303
- Ku, S., Gretton, A., Macke, J., and Logothetis, N. K. (2008). Comparison of pattern recognition methods in classifying high-resolution BOLD signals obtained at high magnetic field in monkeys. *Magn. Reson. Imaging* 26, 1007–1014. doi: 10.1016/j.mri.2008.02.016
- Lemm, S., Blankertz, B., Dickhaus, T., and Müller, K.-R. (2011). Introduction to machine learning for brain imaging. *Neuroimage* 56, 387–399. doi: 10.1016/j.neuroimage.2010.11.004
- Liu, H., and Motoda, H. (1998). *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Boston, MA: Kluwer Academic.
- Logothetis, N. K., Pauls, J., Augath, M., Trinath, T., and Oeltermann, A. (2001). Neurophysiological investigation of the basis of the fMRI signal. *Nature* 412, 150–157. doi: 10.1038/35084005
- Ludwig, K., Kathmann, N., Sterzer, P., and Hesselmann, G. (2015). Investigating category- and shape-selective neural processing in ventral and dorsal visual stream under interocular suppression. *Hum. Brain Mapp.* 36, 137–149. doi: 10.1002/hbm.22618
- Misaki, M., Kim, Y., Bandettini, P. A., and Kriegeskorte, N. (2010). Comparison of multivariate classifiers and response normalizations for pattern-information fMRI. *Neuroimage* 53, 103–118. doi: 10.1016/j.neuroimage.2010.05.051
- Mourão-Miranda, J., Friston, K. J., and Brammer, M. (2007). Dynamic discrimination analysis: a spatial-temporal SVM. *Neuroimage* 36, 88–99. doi: 10.1016/j.neuroimage.2007.02.020
- Mourão-Miranda, J., Reynaud, E., McGlone, F., Calvert, G., and Brammer, M. (2006). The impact of temporal compression and space selection on SVM analysis of single-subject and multi-subject fMRI data. *Neuroimage* 33, 1055–1065. doi: 10.1016/j.neuroimage.2006.08.016
- Müller, K.-R., Mika, S., Ratsch, G., Tsuda, K., and Scholkopf, B. (2001). An introduction to kernel-based learning algorithms. *Neural Netw. IEEE Trans.* 12, 181–201. doi: 10.1109/72.914517
- Mumford, J. A., Turner, B. O., Ashby, F. G., and Poldrack, R. A. (2012). Deconvolving BOLD activation in event-related designs for multivoxel pattern classification analyses. *Neuroimage* 59, 2636–2643. doi: 10.1016/j.neuroimage.2011.08.076
- Naselaris, T., Kay, K. N., Nishimoto, S., and Gallant, J. L. (2011). Encoding and decoding in fMRI. *Neuroimage* 56, 400–410. doi: 10.1016/j.neuroimage.2010.07.073
- Nestor, A., Plaut, D. C., and Behrmann, M. (2011). Unraveling the distributed neural code of facial identity through spatiotemporal pattern analysis. *Proc. Natl. Acad. Sci. U.S.A.* 108, 9998–10003. doi: 10.1073/pnas.1102433108
- Nichols, T. E., and Holmes, A. P. (2002). Nonparametric permutation tests for functional neuroimaging: a primer with examples. *Hum. Brain Mapp.* 15, 1–25. doi: 10.1002/hbm.1058
- Nili, H., Wingfield, C., Walther, A., Su, L., Marslen-Wilson, W., and Kriegeskorte, N. (2014). A toolbox for representational similarity analysis. *PLOS Comput. Biol.* 10:e1003553. doi: 10.1371/journal.pcbi.1003553
- Noirhomme, Q., Lesenfants, D., Gomez, F., Soddu, A., Schrouff, J., Garraux, G., et al. (2014). Biased binomial assessment of cross-validated estimation of classification accuracies illustrated in diagnosis predictions. *Neuroimage Clin.* 4, 687–694. doi: 10.1016/j.nicl.2014.04.004
- Norman, K. A., Polyn, S. M., Detre, G. J., and Haxby, J. V. (2006). Beyond mind-reading: multi-voxel pattern analysis of fMRI data. *Trends Cogn. Sci.* 10, 424–430. doi: 10.1016/j.tics.2006.07.005
- Pereira, F., Mitchell, T., and Botvinick, M. (2009). Machine learning classifiers and fMRI: a tutorial overview. *Neuroimage* 45, S199–S209. doi: 10.1016/j.neuroimage.2008.11.007
- Reverberi, C., Görgen, K., and Haynes, J.-D. (2012). Distributed representations of rule identity and rule order in human frontal cortex and striatum. *J. Neurosci.* 32, 17420–17430. doi: 10.1523/JNEUROSCI.2344-12.2012
- Ritter, C., Hebart, M. N., Wolbers, T., and Bingel, U. (2014). Representation of spatial information in key areas of the descending pain modulatory system. *J. Neurosci.* 34, 4634–4639. doi: 10.1523/JNEUROSCI.4342-13.2014
- Rorden, C. (2007). *MRICron*. Available online at: <http://www.mccauslandcenter.sc.edu/mricro/mricron/>
- Schölkopf, B., Burgess, C., and Vapnik, V. (1995). “Extracting support data for a given task,” in *Proceedings, First International Conference on Knowledge Discovery & Data Mining*. Menlo Park, CA: AAAI Press.
- Schreiber, K., and Kregelberg, B. (2013). The Statistical Analysis of Multi-Voxel Patterns in Functional Imaging. *PLoS ONE* 8:e69328. doi: 10.1371/journal.pone.0069328
- Schrouff, J., Rosa, M. J., Rondina, J. M., Marquand, A. F., Chu, C., Ashburner, J., et al. (2013). PRoNTTo: pattern recognition for neuroimaging toolbox. *Neuroinformatics* 11, 319–337. doi: 10.1007/s12021-013-9178-1
- Seymour, K., Clifford, C. W., Logothetis, N. K., and Bartels, A. (2009). The coding of color, motion, and their conjunction in the human visual cortex. *Curr. Biol.* 19, 177–183. doi: 10.1016/j.cub.2008.12.050
- Sonnenburg, S., Rätsch, G., Henschel, S., Widmer, C., Behr, J., Zien, A., et al. (2010). The SHOGUN machine learning toolbox. *J. Mach. Learn. Res.* 11, 1799–1802. Available online at: <http://dl.acm.org/citation.cfm?id=1859911>
- Stelzer, J., Chen, Y., and Turner, R. (2013). Statistical inference and multiple testing correction in classification-based multi-voxel pattern analysis (MVPA): random permutations and cluster size control. *Neuroimage* 65, 69–82. doi: 10.1016/j.neuroimage.2012.09.063
- Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Ser. B Methodol.* 58, 267–288.

- Van Kemenade, B. M., Seymour, K., Christophel, T. B., Rothkirch, M., and Sterzer, P. (2014a). Decoding pattern motion information in V1. *Cortex* 57, 177–187. doi: 10.1016/j.cortex.2014.04.014
- Van Kemenade, B. M., Seymour, K., Wacker, E., Spitzer, B., Blankenburg, F., and Sterzer, P. (2014b). Tactile and visual motion direction processing in hMT+/V5. *Neuroimage* 84, 420–427. doi: 10.1016/j.neuroimage.2013.09.004
- Weygandt, M., Hackmack, K., Pfüller, C., Bellmann–Strobl, J., Paul, F., Zipp, F., et al. (2011). MRI pattern recognition in multiple sclerosis normal-appearing brain areas. *PLoS ONE* 6:e21138. doi: 10.1371/journal.pone.0021138

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Received: 01 July 2014; accepted: 10 December 2014; published online: 06 January 2015.

Citation: Hebart MN, Görgen K and Haynes J-D (2015) The Decoding Toolbox (TDT): a versatile software package for multivariate analyses of functional imaging data. Front. Neuroinform. 8:88. doi: 10.3389/fninf.2014.00088

This article was submitted to the journal Frontiers in Neuroinformatics.

Copyright © 2015 Hebart, Görgen and Haynes. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) or licensor are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.