NIH Public Access
Author Manuscript
Neuroinformatics. Author manuscript; available in PMC 2009 April 2.

Published in final edited form as:
Neuroinformatics. 2009 ; 7(1): 37–53. doi:10.1007/s12021-008-9041-y.

PyMVPA: A Python toolbox for multivariate pattern analysis of fMRI data

Michael Hanke[*,1,2], **Yaroslav O. Halchenko**[4,5], **Per B. Sederberg**[7,8], **Stephen José Hanson**[4,6], **James V. Haxby**[9,10], and **Stefan Pollmann**[1,2,3]

1*Department of Experimental Psychology, University of Magdeburg, Magdeburg, Germany*

2*Center for Advanced Imaging, Magdeburg, Germany*

3*Center for Behavioral Brain Sciences, Magdeburg, Germany*

4*Psychology Department, Rutgers Newark, New Jersey, USA*

5*Computer Science Department, New Jersey Institute of Technology, Newark, New Jersey, USA*

6*Rutgers University Mind Brain Analysis, Rutgers Newark, New Jersey, USA*

7*Department of Psychology, Princeton University, Princeton, New Jersey, USA*

8*Center for the Study of Brain, Mind, and Behavior, Princeton University, Princeton, New Jersey, USA*

9*Center for Cognitive Neuroscience, Dartmouth College, Hanover, New Hampshire, USA*

10*Department of Psychological and Brain Sciences, Dartmouth College, Hanover, New Hampshire, USA*

Abstract

Decoding patterns of neural activity onto cognitive states is one of the central goals of functional brain imaging. Standard univariate fMRI analysis methods, which correlate cognitive and perceptual function with the blood oxygenation-level dependent (BOLD) signal, have proven successful in identifying anatomical regions based on signal increases during cognitive and perceptual tasks. Recently, researchers have begun to explore new multivariate techniques that have proven to be more flexible, more reliable, and more sensitive than standard univariate analysis. Drawing on the field of statistical learning theory, these new classifier-based analysis techniques possess explanatory power that could provide new insights into the functional properties of the brain. However, unlike the wealth of software packages for univariate analyses, there are few packages that facilitate multivariate pattern classification analyses of fMRI data. Here we introduce a Python-based, cross-platform, and open-source software toolbox, called PyMVPA, for the application of classifier-based analysis techniques to fMRI datasets. PyMVPA makes use of Python's ability to access libraries written in a large variety of programming languages and computing environments to interface with the wealth of existing machine-learning packages. We present the framework in this paper and provide illustrative examples on its usage, features, and programmability.

1. Introduction

Recently, neuroscientists have reported surprising results when they applied machine-learning techniques based on statistical learning theory in their analysis of fMRI data[1]. For example, two such classifier-based studies by Haynes and Rees (2005) and Kamitani and Tong (2005)

*Corresponding Author: Michael Hanke, Otto-von-Guericke-Universität Magdeburg, Institut für Psychologie II, PF 4120, D-39016 Magdeburg, Germany, email: michael.hanke@ovgu.de.

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

were able to predict the orientation of visual stimuli from fMRI data recorded in human primary visual cortex. These studies aggregated information contained in variable, subtle response biases in large numbers of voxels which would not be detected by univariate analysis. These small signal biases were sufficient to disambiguate stimulus orientations despite the fact that their fMRI data were recorded at 3 mm spatial resolution. This is especially notable because the organisation of the primary visual cortex in monkeys indicates that the orientation-selective columns are only approximately 0.5 mm in diameter (Vanduffel et al., 2002), consequently any individual voxel carries only a small amount discriminating information on its own.

Other classifier-based studies have further highlighted the strength of a multivariate analysis approach. For example, classifier-based analysis was first used to investigate neural representations of faces and objects in ventral temporal cortex and showed that the representations of different object categories are spatially distributed and overlapping and revealed that they have a similarity structure that is related to stimulus properties and semantic relationships (Haxby et al., 2001; Hanson et al., 2004; O'Toole et al., 2005).

Nonetheless, the fact that conventional GLM-based analyses are perfectly suitable for a broad range of research topics and, they have another important advantage over classifier-based methods: accessibility. At present there are a large number of well-tested, sophisticated software packages readily available that implement the GLM-based analysis approach. Most of these packages come with convenient graphical and command line interfaces and no longer require profound knowledge of low-level programming languages. This allows researchers to concentrate on designing experiments and to address actual research questions without having to develop specialized analysis scripts for each experiment.

At the time of this writing the authors are aware of only two freely available software packages designed for classifier-based analyses of fMRI data. One is the *3dsvm* plugin for AFNI (LaConte et al., 2005) and the other is the Matlab-based *MVPA toolbox* (Detre et al., 2006). However, both packages only cover a fraction of the available algorithms that have been developed in machine learning research (see NIPS[2] community) over the past decades. For example, the recently founded *Machine learning open source software*[3] project shows an impressive, nonetheless still incomplete, sample of available software packages. At the very least starting with these already available high-quality software libraries has the potential to accelerate scientific progress in the emerging field of classifier-based analysis of brain-imaging data. Although these libraries are freely available, their usage typically assumes a high-level of programming expertise and statistical or mathematical knowledge. Therefore, it would be of great value to have a unifying framework that helps to bridge well-established neuroimaging tools and machine learning software packages and provides ease of programmability, cross library integration and transparent fMRI data handling. The authors propose that such a framework should at least have the five following features:

User-centered programmability with an intuitive user interface Since most neuroimaging researchers are not also computer scientists, it should require only a minimal amount of programming ability. Workflows for typical analyses should be supported by a high-level interface that is focused on the experimental design and language of the neuroimaging scientist. That being said, of course, all interfaces should allow access to detailed information about the internal processing for comprehensive extensibility. Finally, reasonable documentation is a primary requirement.

[1]In the literature, authors have referred to the application of machine-learning techniques to neural data as *decoding* (Kamitani and Tong, 2005; Haynes et al., 2007), *information-based* analysis (*e.g.*Kriegeskorte et al., 2006) or *multi-voxel pattern analysis* (*e.g.*Norman et al., 2006). Throughout this article we will use the term *classifier-based* analysis to refer to all these methods.
[2]Neural Information Processing Systems http://nips.cc/
[3]http://www.mloss.org

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

Extensibility It should be easy to add support for additional external machine learning toolboxes to prevent duplicating the effort that is necessary when a single algorithm has to be implemented multiple times.

Transparent reading and writing of datasets Because the toolbox is focused on neuroimaging data, the default access to data, should require little or no specification for the user. The toolbox framework should also take care of proper conversions into any target data format required for the external machine learning algorithms.

Portability It should not impose restrictions about hardware platforms and should be able to run on all major operating systems.

Open source software It should be open source software, as it allows one to access and to investigate every detail of an implementation, which improves the reproducibility of experimental results, leadings to more efficient debugging and gives rise to accelerated scientific progress (Sonnenburg et al., 2007).

Here we present PyMVPA, a Python-based toolbox for multivariate pattern analysis of fMRI data, which we believe meets all the above criteria for a classifier-based analysis framework. Designed with neuroimaging data in mind, PyMVPA is open-source software that is freely available as source and in binary form from the project website[4].

2. Overview of the Software

PyMVPA is a modular toolbox that basically consists of three components: dataset handling, machine learning algorithms and high-level workflow abstractions. Each module provides an interfaces that connects the toolbox with a large variety of existing software packages (Fig. 1). In the following sections the interfaces to neuro-imaging and machine learning software, and how the three components parts combine to create complete analyses are discussed.

2.1. Bridging fMRI data and machine learning software

Although there are many fMRI data formats, over the last decade the neuroimaging community has converged on *NIfTI* as a standard data format that most fMRI analysis packages support. Thus it was an obvious choice for the primary data storage format supported by PyMVPA.

The situation on the machine learning side, however, is more ambiguous. While specific data formats are of lesser importance here, the variety of programming languages used to develop machine-learning libraries is the main challenge. To this end, the authors selected the *Python*[5] language for PyMVPA because it provides an ideal environment and conforms to the set of features we put forth earlier as critical for this kind of project. Python is a free and open-source scripting language and is available for all major platforms and operating systems. With the project *Neuroimaging in Python* (NIPY; Millman and Brett, 2007)) there is already an ongoing effort to provide a comprehensive software library for traditional fMRI data analysis. In addition, *PyNIfTI*[6] makes it possible to read and write NIfTI files from within the PyMVPA framework and the IPython project provides a powerful Matlab-like command-line interface for interactive data exploration (Perez and Granger, 2007).

With respect to interfacing to pre-existing software packages written in other languages, Python provides PyMVPA with a number of noteworthy features. The *RPy*[7] module allows PyMVPA scripts to make use of the full functionality of the statistical programming language *R*[8] and all

4http://www.pymvpa.org
5http://www.python.org
6http://niftilib.sourceforge.net/pynifti
7http://rpy.sourceforge.net

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

its extensions and support packages. Also, *pymat*[9] and *mlabwrap*[10] provide a similar interface for easy access to Matlab.

Furthermore, Python extensions make it easy to wrap high-performance libraries written in low-level programming languages like C, C++, or Fortran while preserving their speed. However, the *NumPy*[11] and *SciPy*[12] Python packages already provide a fast n-dimensional array library with comprehensive signal processing toolboxes.

In addition to its technical advantages, Python is a well documented, easy to learn, interpreted scripting language, which is instrumental in making PyMVPA an easy and powerful multivariate analysis framework.

2.2. Dataset handling

Input, output, and conversion of datasets are a key task for PyMVPA. A dataset representation has to be simple to allow for maximum interoperability with other toolkits, but simultaneously also has to be comprehensive in order to make available as much information as possible to *e.g.* domain-specific analysis algorithms. In PyMVPA a dataset consists of three parts: the *data samples*, *sample attributes* and *dataset attributes*. While the data samples are the actual patterns that shall be used for training or validation, sample attributes hold additional information on a per sample basis (see Fig. 2). First and foremost of these are the labels that index each data sample with a certain experimental condition and, therefore, define the mapping that will be learned by the classifier.

Additionally, it is often necessary to define groups of data samples. For example, when performing a cross-validation it is necessary to have independent training and validation sets. In the case of fMRI data, with its significant forward temporal contamination across the samples, it is mandatory to take actions to ensure this independence by *e.g.* sufficiently separating training and validation datasets in time. This is typically achieved by splitting an experiment into several runs that are recorded separately. In PyMVPA this type of information can be specified by a special *chunks* sample attribute, where each sample is associated with the numerical identifier of its respective data chunk or run (see Fig. 2). However, an arbitrary number of auxiliary sample attributes can be defined in addition to *labels* and *chunks*.

One of the key features of PyMVPA is its ability to read fMRI datasets and transform them into a generic format that makes it easy for other data processing toolboxes to inherit them. Most machine learning software requires data to be represented in a simple two-dimensional *samples* × *features* matrix (see Fig. 2, bottom), however, fMRI datasets are typically four-dimensional. Although it is possible to view each volume as a simple vector of voxels, doing so discards information about the spatial properties of the volume samples. This is a potentially serious disadvantage because in the context of brain imaging, spatial metrics, and especially distance information, are of interest. In addition, some analysis algorithms such as the *multivariate searchlight* (Kriegeskorte et al., 2006) make use of this information when calculating spheres of voxels.

PyMVPA follows a different approach. Each dataset is accompanied by a transformation or mapping algorithm that preserves all required information and stores it as a dataset attribute. These mappers allow for bidirectional transformations from the original data space into the generic 2-D matrix representation and vice versa. In the case of fMRI volumes the mapper

[8]http://www.r-project.org
[9]http://claymore.engineer.gvsu.edu/~steriana/Python/pymat.html
[10]http://mlabwrap.sourceforge.net
[11]http://numpy.scipy.org
[12]http://www.scipy.org

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

indexes each feature with its original coordinate in the volume. It can optionally compute customizable distances (*e.g.* Cartesian) between features by taking the voxel size along all three dimensions into account. Using the mapper in the reverse direction, from generic feature space into original data space makes it easy to visualize analysis results. For example, feature sensitivity maps can be easily projected back into a 3-D volume and visualized similar to a statistical parametric map.

PyMVPA comes with a specialized dataset type for handling import from and export to images in the NIfTI format[13]. It automatically configures an appropriate mapper by reading all necessary information from the NIfTI file header. Upon export, all header information is preserved (including embedded transformation matrices). This makes it very easy to do further processing in any other NIfTI-aware software package, like *AFNI*[14], *BrainVoyager*[15], *FSL*[16] or *SPM*[17].

Since many algorithms are applied only to a subset of voxels, PyMVPA provides convenient methods to select voxels based on ROI masks. Successively applied feature selections will be taken into account by the mapping algorithm of NIfTI datasets and reverse mappings from the new subspace of features into the original dataspace, *e.g.* for visualization, is automatically performed upon request.

However, the mapper construct in PyMVPA, which is applied to each data sample, is more flexible than a simple 3-D data volume to 1-D feature vector transformation. The original dataspace is not limited to three dimensions. For example, when analyzing an experiment using an event-related paradigm it might be difficult to select a single volume that is representative for some event. A possible solution is to select all volumes covering an event in time, which results in a four-dimensional dataspace. A mapper can also be easily used for EEG/MEG data, *e.g.* mapping spectral decompositions of the time series from multiple electrodes into a single feature vector. PyMVPA provides convenient methods for these use-cases and also supports reverse mapping of results into the original dataspace, which can be of any dimensionality.

2.3. Machine learning algorithms

2.3.1. Classifier abstraction—PyMVPA itself does not at present implement all possible classifiers, even if that were desirable. Currently included are implementations of a k-nearest-neighbor classifier as well as ridge, penalized logistic, Bayesian linear, Gaussian process (GPR), and sparse multinomial logistic regressions (SMLR; Krishnapuram et al., 2005). However, instead of distributing yet another implementation of popular classification algorithms the toolbox defines a generic classifier interface that makes it possible to easily create software wrappers for existing machine learning libraries and enable their classifiers to be used within the PyMVPA framework. At the time of this writing, wrappers for support vector machine algorithms (SVM; Vapnik, 1995) of the widely used *LIBSVM* package (Chang and Lin, 2001) and *Shogun* machine learning toolbox (Sonnenburg et al., 2006) are included. Additional classifiers implemented in the statistical programming language R are provided within PyMVPA (*e.g.* least angle regression, LARS, Efron et al., 2004). The software wrappers expose as much functionality of the underlying implementation as necessary to allow for a seamless integration of the classification algorithm into PyMVPA. Wrapped classifiers can be treated and behave exactly as any of the native implementations.

[13]ANALYZE format is supported as well but it is inferior to NIfTI thus is not explicitly advertised here.
[14]http://afni.nimh.nih.gov/afni
[15]http://www.brainvoyager.com
[16]http://www.fmrib.ox.ac.uk/fsl
[17]http://www.fil.ion.ucl.ac.uk/spm

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

Some classifiers have specific requirements about the datasets they can be trained on. For example, support vector machines (Vapnik, 1995) do not provide native support for multi-class problems, *i.e.* discrimination of more than two classes. To deal with this fact, PyMVPA provides a framework to create meta-classifiers (see Fig. 1). These are classifiers that utilize several basic classifiers, both those implemented in PyMVPA and those from external resources, that are each trained separately and their respective predictions are used to form a joint meta-prediction, sometimes referred to as *boosting* (see Schapire, 2003). Besides generic multi-class support, PyMVPA provides a number of additional meta-classifiers *e.g.* a classifier that automatically applies a customizable feature selection procedure prior to training and prediction. Another example is a meta-classfier that applies an arbitrary mapping algorithm to the data to implement a data reduction step, such as, principal component analysis (PCA), independent component analysis (ICA), both using implementations from *MDP*[18] or wavelet decomposition via *pywavelets*[19].

Despite its high-level interface PyMVPA offers detailed information to the user. To achieve a useful level of transparency, all classifiers can easily store any amount of additional information. For example, a logistic regression might optionally store the output values of the regression that are used to make a prediction. PyMVPA provides a framework to store and pass this information to the user if it is requested. The type and size of such information is in no way limited. However, if the use of additional computational or storage resources is not required, then it can be switched off at any time, to allow for an optimal tradeoff between transparency and performance.

2.3.2. Feature measures—A primary goal for brain-mapping research is to determine where in the brain certain types of information are processed or which regions are engaged in a specific task. In univariate analysis procedures the localization of information is automatically achieved because each feature is tested independently of all others. In contrast, classifier-based analyses, however, incorporates information from the entire feature set to determine whether or not a classifier can extract sufficient information in order to predict the experimental condition from the recorded brain activity. Although classifiers can use the joint signal of the whole feature set to perform their predictions, it is nevertheless important to know which features contribute to the classifiers correct predictions. Some classifiers readily provide information about *sensitivities*, *i.e.* feature-wise scores measuring the impact of each feature on the decision made by the classifier. For example, a simple artificial neural network or a logistic regression, such as SMLR, bases its decisions on a weighted sum of the inputs. Similar weights can also be extracted from any linear classifier including SVMs.

However, there are also *classifier-independent algorithms* to compute featurewise measures. While neural network and SVM weights are inherently multivariate, a feature-wise ANOVA, *i.e.* the fraction of within-class and across class variances, is a univariate measure, as is simple variance or entropy measures of each voxel over all classes. In addition to a simple ANOVA measure PyMVPA provides linear SVM, GPR, and SMLR weights as basic feature sensitivities. As with the classifiers discussed in the previous section, a simple and intuitive interface makes it easy to extend PyMVPA with custom measures (*e.g.* information entropy). Among others, the SciPy package provides a large variety of measures that can be easily used within the PyMVPA framework.

PyMVPA provides some algorithms that can be used on top of the basic featurewise measures to potentially increase their reliability. Multiple feature measures can be easily computed for sub-splits of the training data and combined into a single featurewise measure by averaging,

[18]http://mdp-toolkit.sourceforge.net
[19]http://www.pybytes.com/pywavelets/

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

t-scoring or rank-averaging across all splits. This might help to stabilize measure estimates if a dataset contains spatially distributed artifacts. While a GLM is rather insensitive to such artifacts as it looks at each voxel individually (Chen et al., 2006), classifiers usually pick up such signal if it is related to the classification decision. But, if the artifacts are not equally distributed across the entire experiment, computing measures for separate sub-splits of the dataset can help to identify and reduce their impact on the final measure.

In addition, PyMVPA enables researchers to easily conduct *noise perturbation* analyses, where one measure of interest, such as cross-validated classifier performance, is computed many times with a certain amount of noise added to each feature in turn. Feature sensitivity is then expressed in terms of the difference between computed measures with and without noise added to a feature (see Rakotomamonjy, 2003; Hanson et al., 2004, for equivalence analyses between noise perturbation and simple sensitivities for SVM).

2.4. Workflow abstraction

Classifier-based analyses typically consist of some basic procedures that are independent of the classification algorithm or decision process that was actually used, *e.g.* error calculation, cross-validation of prediction performance, and feature selection. PyMVPA provides support for all of these procedures and, to maximize flexibility, it allows for arbitrary combinations of procedures with any classifiers, featurewise measures, and feature selectors. The two most important procedures are dataset resampling and feature selection.

2.4.1. Dataset resampling—During a typical classifier-based analysis a particular dataset has to be resampled several times to obtain an unbiased generalization estimate of a specific classifier. In the simplest case, resampling is done via splitting the dataset, so that some part serves as a validation dataset while the remaining dataset is used to train a classifier. This is done multiple times until a stable estimate is achieved or the particular sampling procedure exhausts all possible choices to split the data. Proper splitting of a dataset is very important and might not be obvious due to the aforementioned forward contamination of the hemodynamic response function. If the strict separation of training and validation datasets was violated, all subsequent analyses would be biased because the classifier might have had access to the data against which it will be validated.

PyMVPA provides a number of resampling algorithms. The most generic one is an N-M splitter where *M* out of *N* dataset chunks are chosen as the validation dataset while all others serve as training data until all possible combinations of *M* chunks are drawn. This implementation can be used for leave-one-out cross-validation, but additionally provides functionality that is useful for bootstrapping procedures (Efron and Tibshirani, 1993). Additional splitters produce first-half-second-half or odd-even splits. Each splitter may base its splitting on any sample attribute. Therefore it is possible to split not just into different data chunks but also *e.g.* into pairs of stimulus conditions.

Most algorithms implemented in PyMVPA can be parameterized with a splitter, making them easy to apply within different kinds of splitting or cross-validation procedures. Like with other parts of PyMVPA, it is trivial to add other custom splitters, due to a common interface definition.

The dataset resampling functionality in PyMVPA also eases non-parametric testing of classification and generalization performances via a data randomization approach, *e.g.* Monte Carlo permutation testing (Nichols and Holmes, 2001). By running the same analysis multiple times with permuted dataset labels (independently within each data chunk) it is possible to obtain an estimate of the baseline or chance performance of a classifier or some sensitivity

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

measure. This allows one to estimate statistical significance (in terms of p-values) of the results achieved on the original (non-permuted) dataset.

2.4.2. Feature selection procedures—As mentioned above, featurewise measure maps can easily be computed with a variety of algorithms. However, such maps alone cannot answer the question of which features are necessary or sufficient to perform some classification. Feature selection algorithms address this question by trying to determine the relevant features based on featurewise measure. As such, feature selection can be performed in a data-driven or classifier-driven fashion. In a data-driven selection, features could be chosen according to some criterion such as a statistically significant ANOVA score for the feature given a particular dataset, or a statistically significant t-score of one particular weight across splits. Classifier-driven approaches usually involve a sequence of training and validation actions to determine the feature set which is optimal with respect to some classification error (*e.g.* transfer, inherent leave-one-out, theoretical upper-bound) value. It is important to mention that to perform unbiased feature selection using the classifier-driven approach, the selection has to be carried out without observing the main validation dataset for the classifier.

Among the existing algorithms *incremental feature search* (IFS) and *recursive feature elimination* (RFE, Guyon et al., 2002; Guyon and Elisseeff, 2003) are widely used (*e.g.* Rakotomamonjy, 2003) and both available within PyMVPA. The main differences between these procedures are starting point and direction of feature selection. RFE starts with the full feature set and attempts to remove the least-important features until a stopping criterion is reached. IFS on the other hand starts with an empty feature set and sequentially adds important features until a stopping criterion is reached. The implementations of both algorithms in PyMVPA are very flexible as they can be parameterized with all available basic and meta featurewise measures, and expose any desired amount of the progress and internal state of the computation. In addition, the specifics of the iteration process and the stopping criteria are both fully customizable.

2.5. Example analyses

We will demonstrate the functionality of PyMVPA by running some selected analyses on fMRI data from a single participant (participant 1)[20] from a dataset published by Haxby et al. (2001). This dataset was chosen because, since its first publication, it has been repeatedly reanalyzed (Hanson et al., 2004;O'Toole et al., 2007;Hanson and Halchenko, 2008) and parts of it also serve as an example dataset of the Matlab-based MVPA toolbox (Detre et al., 2006).

The dataset itself consists of 12 runs. In each run, the participant passively viewed greyscale images of eight object categories, grouped in 24 s blocks, separated by rest periods. Each image was shown for 500 ms and followed by a 1500 ms inter-stimulus interval. Full-brain fMRI data were recorded with a volume repetition time of 2500 ms, thus, a stimulus block was covered by roughly 9 volumes. For a complete description of the experimental design and fMRI acquisition parameters see Haxby et al. (2001).

Prior to any analysis, the raw fMRI data were motion corrected using FLIRT from FSL (Jenkinson et al., 2002). All data processing that followed was performed with PyMVPA[21]. After motion correction, linear detrending was performed for each run individually by fitting a straight line to each voxels timeseries and subtracting it from the data. No additional spatial or temporal filtering was applied.

[20]Given that the results reported are from a single participant, we are simply illustrating the capabilities of PyMVPA, not trying to promote any analysis method as more-effective than another.

[21]Note that PyMVPA internally makes use of a number of other aforementioned Python modules, such as NumPy and SciPy.

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

For the sake of simplicity, we focused on the binary CATS vs. SCISSORS classification problem. All volumes recorded during either CATS or SCISSORS blocks were extracted and voxel-wise *Z*-scored with respect to the mean and standard deviation of volumes recorded during rest periods. *Z*-scoring was performed individually for each run to prevent any kind of information transfer across runs.

Every analysis is accompanied by source code snippets that show their implementation using the PyMVPA toolbox. For demonstration purposes they are limited to the most important steps.

2.5.1. Load a dataset—Dataset representation in PyMVPA builds on NumPy arrays. Anything that can be converted into such an array can also be used as a dataset source for PyMVPA. Possible formats range from various plain text formats to binary files. However, the most important input format from the functional imaging perspective is NIfTI[22], which PyMVPA supports with a specialized module.

The following short source code snippet demonstrates how a dataset can be loaded from a NIfTI image. PyMVPA supports reading the sample attributes from a simple two-column text file that contains a line with a label and a chunk id for each volume in the NIfTI image (line 0). To load the data samples from a NIfTI file it is sufficient to create a `NiftiDataset` object with the filename as an argument (line 1). The previously-loaded sample attributes are passed to their respective arguments as well (lines 2-3). Optionally, a mask image can be specified (line 4) to easily select a subset of voxels from each volume based on the non-zero elements of the mask volume. This would typically be a mask image indicating brain and non-brain voxels.

```
0    attr = SampleAttributes ('sample _attr_filename.txt')
1    dataset = NiftiDataset (samples='subj1_bold.nii.gz',
2                            labels=attr.labels,
3                            chunks=attr.chunks,
4                            mask='subj1_roi_mask.nii.gz')
```

Once the dataset is loaded, successive analysis steps, such as feature selection and classification, only involve passing the `dataset` object to different processing objects. All following examples assume that a dataset was already loaded.

2.5.2. Simple full-brain analysis—The first analysis example shows the few steps necessary to run a simple cross-validated classification analysis. After a dataset is loaded, it is sufficient to decide which classifier and type of splitting shall be used for the cross-validation procedure. Everything else is automatically handled by `CrossValidatedTransferError`. The following code snippet performs the desired classification analysis via leave-one-out cross-validation. Error calculation during cross-validation is conveniently performed by `TransferError`, which is configured to use a linear C-SVM classifier[23] on line 6. The leave-one-out cross-validation type is specified on line 7.

```
5    cv = CrossValidatedTransferError (
6         transfer_error=TransferError (LinearCSVMC ()),
```

[22]To a certain degree PyMVPA also supports importing ANALYZE files.
[23]LIBSVM C-SVC (Chang and Lin, 2001) with trade-off parameter *C* being a reciprocal of the squared mean of Frobenius norms of the data samples.

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

```
7        splitter=NFoldSplitter (cvtype=1))
8        mean_error = cv (dataset)
```

Simply passing the dataset to cv (line 8) yields the mean error. The computed error defaults to the fraction of incorrect classifications, but an alternative error function can be passed as an argument to the `TransferError` call. If desired, more detailed information is available, such as a confusion matrix based on all the classifier predictions during cross-validation.

2.5.3. Multivariate searchlight—One method to localize functional information in the brain is to perform a classification analysis in a certain region of interest (ROI; *e.g.* Pessoa and Padmala, 2007). The rationale for size, shape and location of a ROI can be *e.g.* anatomical landmarks or functional properties determined by a GLM-contrast.

Alternatively, Kriegeskorte et al. (2006) proposed an algorithm that scans the whole brain by running multiple ROI analyses. The so-called *multivariate searchlight* runs a classifier-based analysis on spherical ROIs of a given radius centered around any voxel covering brain matter. Running a searchlight analysis computing *e.g.* generalization performances yields a map showing where in the brain a relevant signal can be identified while still harnessing the power of multivariate techniques (for application examples see Haynes et al., 2007; Kriegeskorte et al., 2007).

A searchlight performs well if the target signal is available within a relatively small area. By increasing size of the searchlight the information localization becomes less specific because, due to the anatomical structure of the brain, each spherical ROI will contain a growing mixture of grey-matter, white-matter, and non-brain voxels. Additionally, a searchlight operating on volumetric data will integrate information across brain-areas that are not directly connected to each other *i.e.* located on opposite borders of a sulcus. This problem can be addressed by running a searchlight on data that has been transformed into a surface representation. PyMVPA supports analyses with spatial searchlights (not extending in time), operating on both volumetric and surface data (given an appropriate mapping algorithm and using circular patches instead of spheres). The searchlight implementation can compute an arbitrary measure within each sphere.

In the following example, the measure to be computed by the searchlight is configured first. Similar to the previous example it is a cross-validated transfer or generalization error, but this time it will be computed on an odd-even split of the dataset and with a linear C-SVM classifier (lines 9-11). On line 12 the searchlight is setup to compute this measure in all possible 5 mm-radius spheres when called with a dataset (line 13) The final call on line 14 transforms the computed error map back into the original data space and stores it as a compressed NIfTI file. Such a file can then be viewed and further processed with any NIfTI-aware toolkit.

```
9        cv = CrossValidatedTransferError (
10       transfer_error=TransferError (LinearCSVMC ()),
11       splitter=OddEvenSplitter ())
12       sl = Searchlight (cv, radius=5)
13       sl_map = sl (dataset)
14       dataset.map2Nifti (sl_map).save ('searchlight_5mm.nii.gz')
```

Figure 3 shows the searchlight error maps for the CATS vs. SCISSORS classification on single volumes from the example dataset for radii of 1, 5, 10 and 20 mm respectively. Utilizing only a single voxel in each sphere (1 mm radius), yields a generalization error as low as 17% in the

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

best performing sphere, which is located in the left occipito-temporal fusiform cortex. With increases in the radius there is a tendency for further error reduction, indicating that the classifier performance benefits from integrating signal from multiple voxels. However, better classification accuracy is achieved at the cost of reduced spatial precision of signal localization. The distance between the centers of the best-performing spheres for 5 and 20 mm searchlights totals almost 18 mm. The lowest overall error in the right occipito-temporal cortex with 8% is achieved by a searchlight with a radius of 10 mm. The best performing sphere with 20 mm radius (12% generalization error) is centered between right inferior temporal and fusiform gyrus. It comprises approximately 700 voxels and extends from right lingual gyrus to the right inferior temporal gyrus, also including parts of the cerebellum and left lateral ventricle. It, therefore, includes a significant proportion of voxels sampling cerebrospinal fluid or white matter, indicating that a sphere of this size is not optimal given the structural organization of the brain surface. Kriegeskorte et al. (2006) suggest that a sphere radius of 4 mm yields near-optimal performance. However, while this assumption might be valid for representations of object properties or low-level visual features, a searchlight of this size could miss signals related to high-level cognitive processes that involve several spatially distinct functional subsystems of the brain.

2.5.4. Feature selection—Feature selection is a common preprocessing step that is also routinely performed as part of a conventional fMRI data analysis, *i.e.* the initial removal of non-brain voxels. This basic functionality is provided by `NiftiDataset` as it was shown on line 4 to provide an initial operable feature set. Likewise, a searchlight analysis also involves multiple feature selection steps (*i.e.* ROI analyses), based on the spatial configuration of features. Nevertheless, PyMVPA provides additional means to perform feature selection, which are not specific to the fMRI domain, in a transparent and unified way.

Machine learning algorithms often benefit from the removal of noisy and irrelevant features (see Guyon et al., 2002, Section V.1. “The features selected matter more than the classifier used”). Retaining only features relevant for classification improves learning and generalization of the classifier by reducing the possibility of overfitting the data. Therefore, providing a simple interface to feature selection is critical to gain superior generalization performance and get better insights about the relevance of a subset of features with respect to a given contrast. Table 1 shows the prediction error of a variety of classifiers on the full example dataset with and without any prior feature selection. Most of the classifiers perform near chance performance without prior feature selection[24], and even simple feature selection (*e.g.* some percentage of the population with highest scores on some measure) boosts generalization performance significantly of all classifiers, including the non-linear algorithms radial basis function SVM and, kNN.

PyMVPA provides an easy way to perform feature selections. The `FeatureSelectionClassifier` is a meta-classifier that enhances any classifier with an arbitrary initial feature selection step. This approach is very flexible as the resulting classifier can be used as any other classifier, *e.g.* for unbiased generalization testing using `CrossValidatedTransferError`. For instance, the following example shows a classifier that operates only on 5% of the voxels that have the highest ANOVA score across the data categories in a particular dataset. It is noteworthy that the source code looks almost identical to the example given on lines 5-8, with just the feature selection method added to it. No changes are necessary for the actual cross-validation procedure.

[24]Chance performance without feature selection was not the norm for all category pairs in the dataset. For example, the SVM classifier generalized well for other pairs of categories (*e.g.* FACE vs HOUSE) without prior feature selection. Consequently, SCISSORS vs CATS was chosen to provide a more difficult analysis case.

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

```
15 clf = FeatureSelectionClassifier (
16     clf=LinearCSVMC (),
17     feature_selection=SensitivityBasedFeatureSelection (
18         sensitivity_analyzer=OneWayAnova (),
19         feature_selector=
20             FractionTailSelector (0.05, mode='select', tail='upper'))
21 cv = CrossValidatedTransferError (
22     transfer_error=TransferError (clf),
23     splitter=NFoldSplitter (cvtype=1))
24 mean_error = cv (dataset)
```

It is important to emphasize that feature selection (lines 17-19) in this case is *not* performed first on the full dataset, which could bias generalization estimation. Instead, feature selection is being performed as a part of classifier training, thus, only the actual training dataset is visible to the feature selection. Due to the unified interface, it is possible to create a more sophisticated example, where feature selection is performed via *recursive feature elimination* (Guyon et al., 2002; Guyon and Elisseeff, 2003):

```
25 rfesvm = LinearCSVMC ()
26 clf = SplitClassifier (
27     FeatureSelectionClassifier (
28         clf=rfesvm,
29         feature_selection=RFE (
30             sensitivity_analyzer=
31                 LinearSVMWeights (clf=rfesvm,
32                     transformer=Absolute),
33             transfer_error=TransferError (rfesvm),
34         stopping_criterion=FixedErrorThresholdStopCrit (0.05),
35         feature_selector=
36             FractionTailSelector (0.2, mode='discard', tail='lower'),
37         update_sensitivity=True)),
38     splitter=NFoldSplitter ())
```

On line 25 we define the main classifier that is reused in many aspects of the processing: line 28 specifies that classifier to be used to make the final prediction operating only on the selected features, line 31 instructs the sensitivity analyzer to use it to provide sensitivity estimates of the features at each step of recursive feature elimination, and on line 33 we specify that the error used to select the best feature set is a generalization error of that same classifier. Utilization of the same classifier for both the sensitivity analysis and for the transfer error computation prevents us from re-training a classifier twice for the same dataset.

The fact that the RFE approach is classifier-driven requires us to provide the classifier with two datasets: one to train a classifier and assess its features sensitivities and the other one to determine stopping point of feature elimination based on the transfer error. Therefore, the `FeatureSelectionClassifier` (line 27) is wrapped within a `SplitClassifier` (line 26), which in turn uses `NFoldSplitter` (line 38) to generate a set of data splits on which to train and test each independent classifier. Within each data split, the classifier selects its features

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript

independently using RFE by computing a generalization error estimate (line 33) on the internal validation dataset generated by the splitter. Finally, the `SplitClassifier` uses a customizable voting strategy (by default *MaximalVote*) to derive the joint classification decision.

As before, the resultant classifier can now simply be used within `CrossValidatedTransferError` to obtain an unbiased generalization estimate of the trained classifier. The step of validation onto independent validation dataset is often overlooked by the researchers performing RFE (Guyon et al., 2002). That leads to biased generalization estimates, since otherwise internal feature selection of the classifier is driven by the full dataset.

Fortunately, some machine learning algorithms provide internal theoretical upper bound on the generalization performance, thus they could be used as a `transfer_error` criterion (line 33) with RFE, which eliminates the necessity of additional splitting of the dataset. Some other classifiers perform feature selection internally (*e.g.* SMLR, also see figure 4), which removes the burden of external explicit feature selection and additional data splitting.

3. Discussion

Numerous studies have illustrated the power of classifier-based analyses, harnessing machine-learning techniques based on statistical learning theory to extract information about the functional properties of the brain previously thought to be below the signal-to-noise ratio of fMRI data (for reviews see Haynes and Rees, 2006; Norman et al., 2006).

Although the studies cover a broad range of topics from human memory to visual perception, it is important to note that they were performed by relatively few research groups. This may be due to the fact that very few software packages that specifically address classifier-based analyses of fMRI data are available to a broad audience. Such packages require a significant amount of software development, starting from basic problems, such as how to import and process fMRI datasets, to more complex problems, such as the implementation of classifier algorithms. This results in an initial overhead requiring significant resources before actual neuroimaging datasets can be analyzed.

The PyMVPA toolbox aims to be a solid base for conducting classifier-based analyses. In contrast to the *3dsvm* plugin for AFNI (LaConte et al., 2005), it follows a more general approach by providing a collection of common algorithms and processing steps that can be combined with great flexibility. Consequently, the initial overhead to start an analysis once the fMRI dataset is acquired is significantly reduced because the toolbox also provides all necessary import, export and preprocessing functionality.

PyMVPA is specifically tuned towards fMRI data analysis, but the generic design of the framework allows to work with other data modalities equally well. The flexible dataset handling allows one to easily extend it to other data formats, while at the same time extending the mapping algorithms to represent other data spaces and metrics, such as the sparse surface sampling of EEG channels or MEG datasets (see Thulasidas et al., 2006; Guimaraes et al., 2007, for examples of classifier-based analysis of these data modalities).

However, the key feature of PyMVPA is that it provides a uniform interface to bridge from standard neuroimaging tools to machine learning software packages. This interface makes it easy to extend the toolbox to work with a broad range of *existing* software packages, which should significantly reduce the need to recode available algorithms for the context of brain-imaging research. Moreover, all external and internal classifiers can be freely combined with the classifier-independent algorithms for *e.g.* feature selection, making this toolbox an ideal environment to compare different classification algorithms.

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

The introduction listed portability as one of the goals for an optimal analysis framework. PyMVPA code is tested to be portable across multiple platforms, and limiting set of essential external dependencies in turn has proven to be portable. In fact, PyMVPA only depends on a moderately recent version of Python and NumPy package. Although PyMVPA can make use of other external software, the functionality provided by them is completely optional. For an up-to-date list of possible extensions the reader is referred to the PyMVPA project website[25]. To allow for convenient installation and upgrade procedures, the authors are providing binary packages for ten different operating systems, including various GNU/Linux distributions (in their native package format), as well as installers for MacOS X and Windows. This comprises PyMVPA itself and a number of additional packages (*e.g.* NumPy), if they are not available from other sources for a particular target platform.

Although PyMVPA aims to be especially user-friendly it does not provide a graphical user interface (GUI). The reason not to include such an interface is that the toolbox explicitly aims to encourage novel combinations of algorithms and the development of new analysis strategies that are not easily foreseeable by a GUI designer[26]. The toolbox is nevertheless user-friendly, enabling researchers to conduct highly complex analyses with just a few lines of easily readable code. It achieves this by taking away the burden of dealing with low-level libraries and providing a great variety of algorithms in a concise framework. The required skills of a potential PyMVPA user are not much different from neuroscientists using the basic building blocks needed for one of the established fMRI analysis toolkits (e.g., shell scripting for AFNI and FSL command line tools, or Matlab-scripting of SPM functions).

PyMVPA is an actively developed project. Further releases will significantly extend the currently available functionality. The authors are continuously looking for machine learning algorithms and toolboxes that provide features that are interesting in the context of neuroimaging. For example, we recently added support for sparse multinomial logistic regression, which shows great promise as both a multivariate feature selection tool and a classifier (Krishnapuram et al., 2005).

Recent releases of PyMVPA added support for visualization of analysis results, such as, classifier confusions, distance matrices, topography plots and plotting of time-locked signals. However, while PyMVPA does not provide extensive plotting support it nevertheless makes it easy to use existing tools for MRI-specific data visualization. Similar to the data import PyMVPA's mappers make it also trivial to export data into the original data space and format, *e.g.* using a reverse-mapped sensitivity volume as a statistical overlay, in exactly the same way as a statistical parametric map (SPM) derived from a conventional analysis. This way PyMVPA can fully benefit from the functionality provided by the numerous available MRI toolkits.

Another topic of future development is information localization. Chen et al. (2006) recently emphasized that, especially in the context of brain mapping, it is important not to focus only on prediction accuracies, but also to examine the stability of the feature selections within a cross-validation. Given that a typical standard-resolution full-brain fMRI dataset contains roughly 30–50 thousand features it might be perfectly possible that any individual classification can be performed with close-to-perfect accuracy, but within each cross-validation fold a completely non-overlapping set of voxels is chosen. Figure 4 shows examples of ratios of cross-validation folds in which any given feature was chosen by the corresponding feature selection method used by some classifiers listed in Table 1. Such variability in the selected features might be due to the scenario when the signal from all unique feature sets is redundant and the feature

[25]http://www.pymvpa.org/installation.html#dependencies
[26]Nothing prevents a software developer from adding a GUI to the toolbox using one of the many GUI toolkits that interface with Python code, such as PyQT (http://www.riverbankcomputing.co.uk/pyqt/) or wxPython (http://www.wxpython.org/).

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

selections differ simply due to random variations in the noise pattern. PyMVPA already provides convenient methods to assess the stability of feature selections within cross-validation procedures. However, more research is required to address information localization problems in different contexts. For example, when implementing a brain-computer interface it is beneficial to identify a set of features that provides both an optimal generalization performance as well as a high stability of spatial configuration and accuracy across different datasets, *i.e.* to reduce the number of false-positive feature selections. On the other hand, in a clinical setting one may want to identify all voxels that could possibly contribute some information in a pre-surgery diagnostic tool and, thus, would focus on minimizing false-negatives instead.

The features of PyMVPA outlined here cover only a fraction of the currently implemented functionality. This article focuses on the analysis of fMRI and therefore does not elaborate on the possibilities of *e.g.* multi-modal and non-fMRI data analysis, as well as the possibility for the analysis of event-related paradigms. More information is, however, available on the PyMVPA project website, which contains user manual with an introduction into the main concepts and the design of the framework, a wide range of examples, a comprehensive module reference as a user-oriented summary of the available functionality, and finally a more technical reference for extending the framework.

The emerging field of classifier-based analysis of fMRI data is beginning to complement the established analysis techniques and has great potential for novel insights into the functional architecture of the brain. However, there are a lot of open questions how the wealth of algorithms developed by those motivated by statistical learning theory can be optimally applied to brain-imaging data (*e.g.* fMRI-aware feature selection algorithms or statistical inference of classifier performances). The lack of a *gold standard* for classifier-based analysis demands software that allows one to apply a broad range of available techniques and test an even broader range of hypotheses. PyMVPA tries to reach this goal by providing a unifying framework that allows to easily combine a large number of basic building blocks in a flexible manner to help neuroscientists to do rapid initial data exploration and, consecutive custom data analysis. PyMVPA even facilitates the integration of additional algorithms in its framework that are not yet discovered by neuro-imaging researchers. Despite being able to perform complex analyses, PyMVPA provides a straightforward programming interface based on an intuitive scripting language. The availability of more user-friendly tools, like PyMVPA, will hopefully attract more researchers to conduct classifier-based analyses and, thus, explore the full potential of statistical learning theory based techniques for brain-imaging research.

4. Information Sharing Statement

The PyMVPA toolbox is free and open source software, and is available from the project website (http://www.pymvpa.org). FMRI dataset used for the example analysis in the paper is available from the same website.

Acknowledgements

Michael Hanke and Stefan Pollmann were supported by the German Academic Exchange Service (Grant: PPP-USA D/05/504/7). Per Sederberg was supported by National Institutes of Health NRSA grant MH080526. Yaroslav O. Halchenko and Dr. Stephen J. Hanson were supported by National Science Foundation (grant: SBE 0751008) and James McDonnell Foundation (grant: 220020127).

References

Chang CC, Lin CJ. LIBSVM: a library for support vector machines. 2001Software available at

Chen X, Pereira F, Lee W, Strother S, Mitchell T. Exploring predictive and reproducible modeling with the single-subject FIAC dataset. Human Brain Mapping 2006;27:452–461. [PubMed: 16565951]

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

Detre, G.; Polyn, SM.; Moore, C.; Natu, V.; Singer, B.; Cohen, J.; Haxby, JV.; Norman, KA. The multi-voxel pattern analysis (MVPA) toolbox. Poster presented at the Annual Meeting of the Organization for Human Brain Mapping; Florence, Italy. 2006.

Efron, B.; Tibshirani, R. An Introduction to the Bootstrap. Chapman & Hall/CRC; 1993.

Efron B, Trevor H, Johnstone I, Tibshirani R. Least angle regression. Annals of Statistics 2004;32:407–499.

Guimaraes MP, Wong DK, Uy ET, Grosenick L, Suppes P. Single-trial classification of MEG recordings. IEEE Transactions on Biomedical Engineering 2007;54:436–443. [PubMed: 17355055]

Guyon I, Elisseeff A. An introduction to variable and feature selection. Journal of Machine Learning 2003;3:1157–1182.

Guyon I, Weston J, Barnhill S, Vapnik V. Gene selection for cancer classification using support vector machines. Machine Learning 2002;46:389–422.

Hanson S, Matsuka T, Haxby J. Combinatorial codes in ventral temporal lobe for object recognition: Haxby (2001) revisited: is there a "face" area? Neuroimage 2004;23:156–166. [PubMed: 15325362]

Hanson SJ, Halchenko YO. Brain reading using full brain support vector machines for object recognition: there is no "face" identification area. Neural Computation 2008;20:486–503. [PubMed: 18047411]

Haxby J, Gobbini M, Furey M, Ishai A, Schouten J, Pietrini P. Distributed and overlapping representations of faces and objects in ventral temporal cortex. Science 2001;293:2425–2430. [PubMed: 11577229]

Haynes JD, Rees G. Predicting the orientation of invisible stimuli from activity in human primary cortex. Nature Neuroscience 2005;8:686–691.

Haynes JD, Rees G. Decoding mental states from brain activity in humans. Nature Reviews Neuroscience 2006;7:523–534.

Haynes JD, Sakai K, Rees G, Gilbert S, Frith C, Passingham RE. Reading hidden intentions in the human brain. Current Biology 2007;17:323–328. [PubMed: 17291759]

Jenkinson M, Bannister P, Brady J, Smith S. Improved optimisation for the robust and accurate linear registration and motion correction of brain images. Neuroimage 2002;17:825–841. [PubMed: 12377157]

Kamitani Y, Tong F. Decoding the visual and subjective contents of the human brain. Nature Neuroscience 2005;8:679–685.

Kriegeskorte N, Formisano E, Sorger B, Goebel R. Individual faces elicit distinct response patterns in human anterior temporal cortex. Proceedings of the National Academy of Sciences of the USA 2007;104:20600–20605. [PubMed: 18077383]

Kriegeskorte N, Goebel R, Bandettini P. Information-based functional brain mapping. Proceedings of the National Academy of Sciences of the USA 2006;103:3863–3868. [PubMed: 16537458]

Krishnapuram B, Carin L, Figueiredo MA, Hartemink AJ. Sparse multinomial logistic regression: fast algorithms and generalization bounds. IEEE Transactions on Pattern Analysis and Machine Intelligence 2005;27:957–968. [PubMed: 15943426]

LaConte S, Strother S, Cherkassky V, Anderson J, Hu X. Support vector machines for temporal classification of block design fmri data. Neuroimage 2005;26:317–329. [PubMed: 15907293]

Millman K, Brett M. Analysis of functional magnetic resonance imaging in python. Computing in Science & Engineering 2007;9:52–55.

Nichols TE, Holmes AP. Nonparametric permutation tests for functional neuroimaging: A primer with examples. Human Brain Mapping 2001;15:1–25. [PubMed: 11747097]

Norman KA, Polyn SM, Detre GJ, Haxby JV. Beyond mind-reading: multi-voxel pattern analysis of fmri data. Trends in Cognitive Science 2006;10:424–430.

O'Toole AJ, Jiang F, Abdi H, Haxby JV. Partially distributed representations of objects and faces in ventral temporal cortex. Journal of Cognitive Neuroscience 2005;17:580–590. [PubMed: 15829079]

O'Toole AJ, Jiang F, Abdi H, Penard N, Dunlop JP, Parent MA. Theoretical, statistical, and practical perspectives on pattern-based classification approaches to the analysis of functional neuroimaging data. Journal of Cognitive Neuroscience 2007;19:1735–1752. [PubMed: 17958478]

Perez F, Granger B. IPython: A system for interactive scientific computing. Computing in Science & Engineering 2007;9:21–29.

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

Pessoa L, Padmala S. Decoding near-threshold perception of fear from distributed single-trial brain activation. Cerebral Cortex 2007;17:691–701. [PubMed: 16627856]

Rakotomamonjy A. Variable selection using SVM-based criteria. Journal of Machine Learning Research 2003;3:1357–1370.

Schapire, RE. The boosting approach to machine learning: An overview. In: Denison, DD.; Hansen, MH.; Holmes, C.; Mallick, B.; Yu, B., editors. Nonlinear Estimation and Classification. Springer; New York: 2003.

Sonnenburg S, Braun M, Ong CS, Bengio S, Bottou L, Holmes G, LeCun Y, Müller KR, Pereira F, Rasmussen CE, Rätsch G, Schölkopf B, Smola A, Vincent P, Weston J, Williamson R. The need for open source software in machine learning. Journal of Machine Learning Research 2007;8:2443–2466.

Sonnenburg S, Raetsch G, Schaefer C, Schoelkopf B. Large scale multiple kernel learning. Journal of Machine Learning Research 2006;7:1531–1565.

Thulasidas M, Guan C, Wu J. Robust classification of eeg signal for brain-computer interface. IEEE Transactions on Neural Systems and Rehabilitation Engineering 2006;14:24–29. [PubMed: 16562628]

Vanduffel W, Tootell RBH, Schoups AA, Orban GA. The organization of orientation selectivity throughout macaque visual cortex. Cerebral Cortex 2002;12:647–662. [PubMed: 12003864]

Vapnik, V. The Nature of Statistical Learning Theory. Springer; New York: 1995.

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

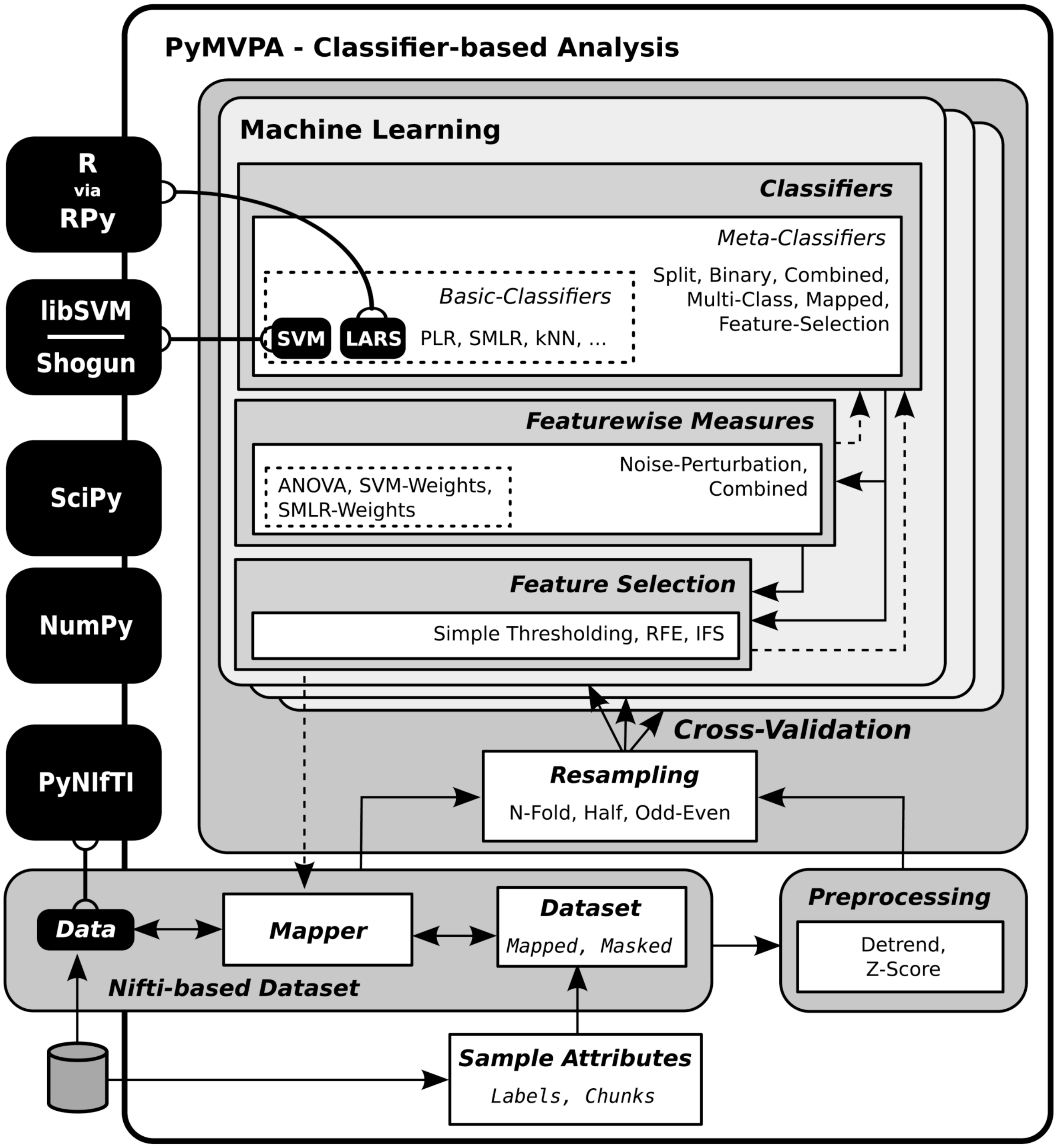

Figure 1.

PyMVPA workflow design. Datasets can be easily loaded from NIfTI files (PyNIfTI) and other sources. The available machine learning algorithms include basic classifiers (also via wrappers from external resources *e.g.* libsvm), basic featurewise measures, and meta-algorithms *e.g.* generic multi-class classifier support and feature selection algorithms such as *recursive feature selection* (RFE, Guyon et al., 2002; Guyon and Elisseeff, 2003). Any analysis built from those basic elements can be cross-validated by running them on multiple dataset splits that can be easily generated with a variety of data resampling procedures (*e.g.* bootstrapping, Efron and Tibshirani, 1993). Due to the simplicity of the PyMVPA datasets every (intermediate) analysis

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript

result is compatible with a broad range of external processing algorithms available from other Python software packages such as *Numerical Python* (NumPy) or *Scientific Python* (SciPy).

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

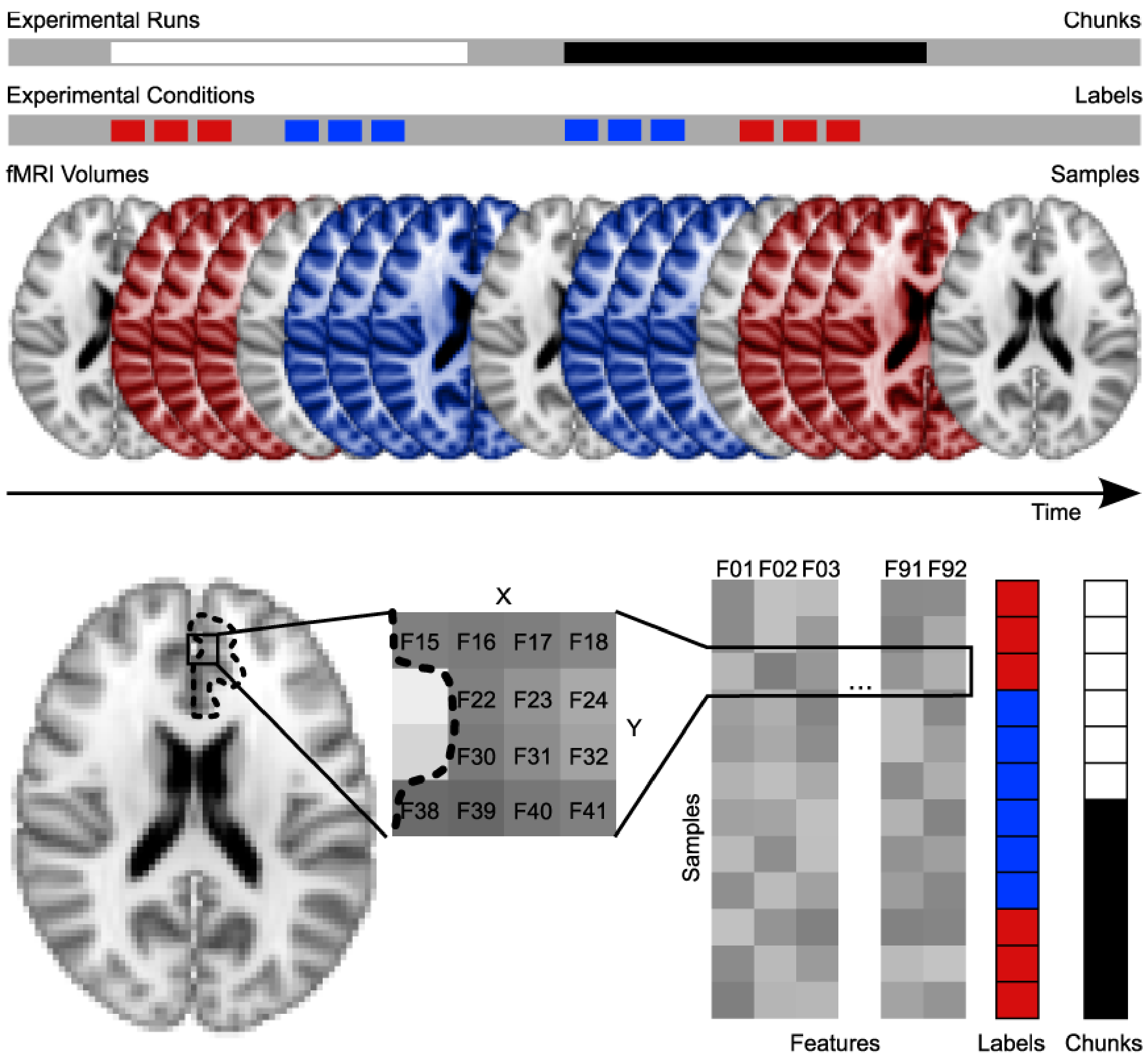

Figure 2.
Terminology for classifier-based analyses in PyMVPA. The upper part shows a simple block-design experiment with two experimental conditions (*red* and *blue*) and two experimental runs (*black* and *white*). Experimental runs are referred to as independent *chunks* of data and fMRI volumes recorded in certain experimental conditions are data *samples* with the corresponding condition *labels* attached to them (for the purpose of visualization the axial slices are taken from the MNI152 template downsampled to 3 mm isotopic resolution). The lower part shows an example ROI analysis of that paradigm. All voxels in the defined ROI are considered as *features*. The three-dimensional data samples are transformed into a two-dimensional *samples*×*feature* representation, where each row (sample) of the data matrix is associated with a certain label and data chunk.

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

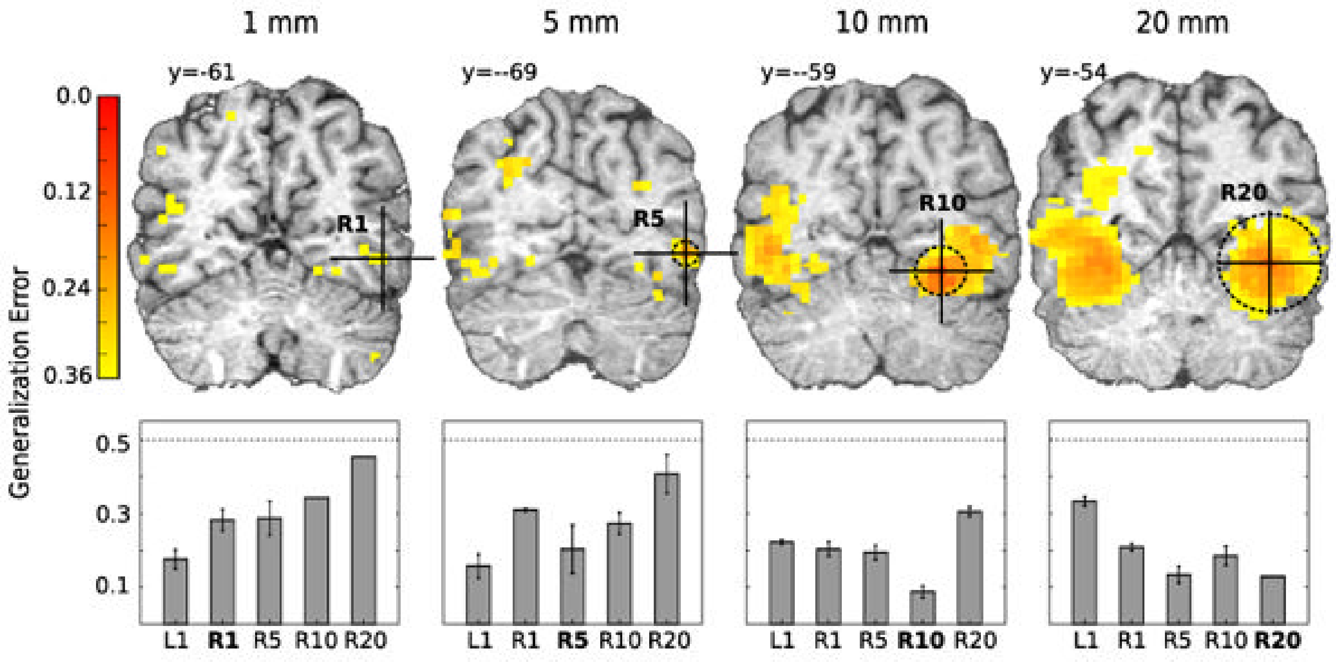

Figure 3.
Searchlight analysis results for CATS vs. SCISSORS classification for sphere radii of 1, 5, 10 and 20 mm (corresponding to approximately 1, 15, 80 and 675 voxels per sphere respectively). The upper part shows generalization error maps for each radius. All error maps are thresholded arbitrarily at 0.35 (chance level: 0.5) and are not smoothed to reflect the true functional resolution. The center of the best performing sphere (*i.e.* lowest generalization error) in right temporal fusiform cortex or right lateral occipital cortex is marked by the cross-hair on each coronal slice. The dashed circle around the center shows the size of the respective sphere (for radius 1 mm the sphere only contains a single voxel). MNI-space coordinates (*x*, *y*, *z*) in mm for the four sphere centers are: 1 mm (*R1*): (48, -61, -6), 5 mm (*R5*): (48, -69, -4), 10 mm (*R10*): (28, -59, -12) and 20 mm (*R20*): (40, -54, -8). The lower part shows the generalization errors for spheres centered around these four coordinates, plus the location of the univariately best performing voxel (*L1*: -35, -43, -23; left occipito-temporal fusiform cortex) for all radii. The error bars show the standard error of the mean across cross-validation folds.

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

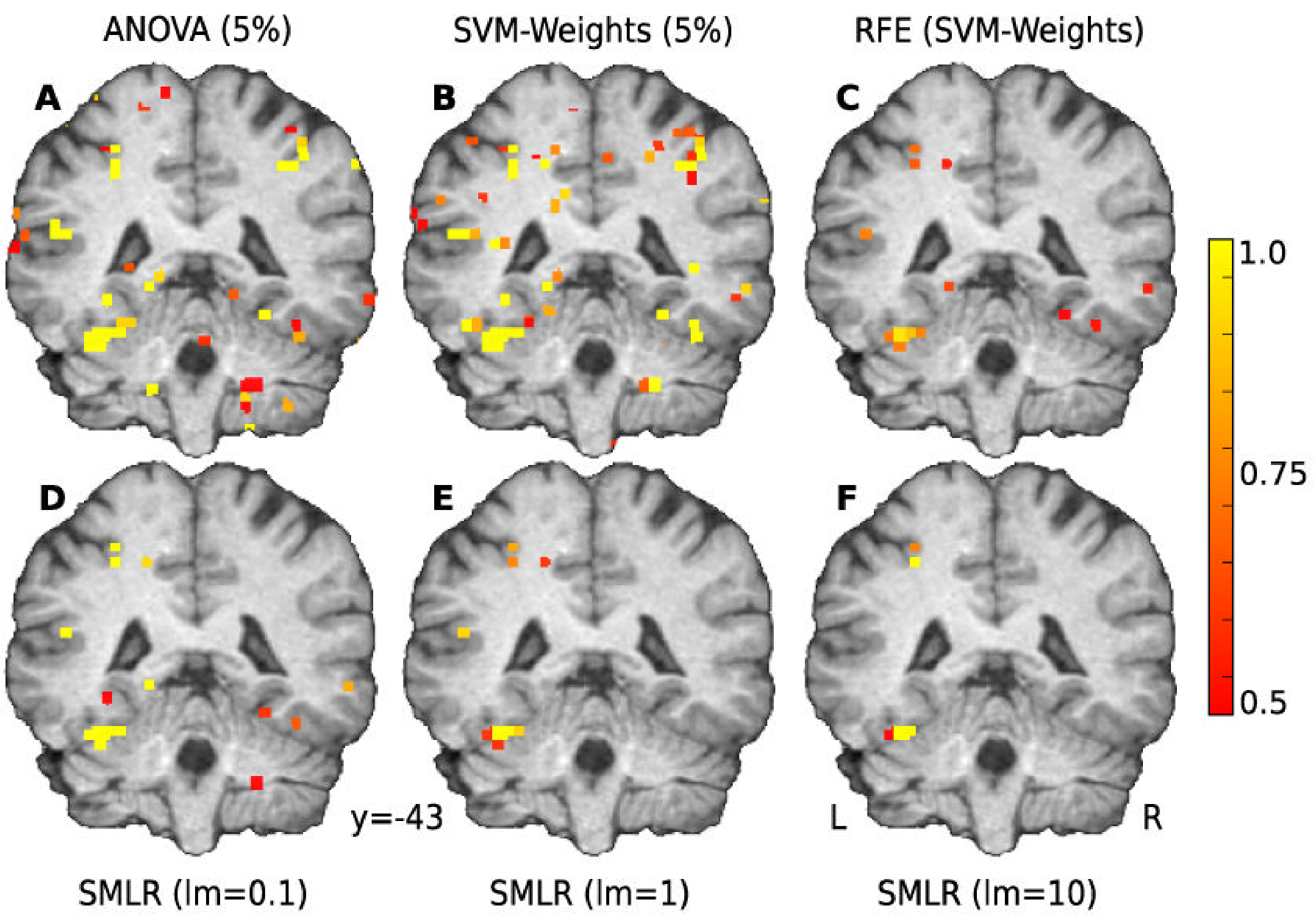

Figure 4.
Feature selection stability maps for the CATS vs. SCISSORS classification. The color maps show the fraction of cross-validation folds in which each particular voxel is selected by various feature selection methods used by the classifiers listed in Table 1: 5% highest ANOVA *F*-scores (A), 5% highest weights from trained SVM (B), RFE using SVM weights as selection criterion (C), internal feature selection performed by the SMLR classifier with penalty term 0.1 (D), 1 (E) and 10 (F). All methods reliably identify a cluster of voxels in the left fusiform cortex, centered around MNI: -34, -43, -20 mm. All stability maps are thresholded arbitrarily at 0.5 (6 out of 12 cross-validation folds).

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

Table 1

Performance of various classifiers with and without feature selection. For classifiers with feature selection the classifier algorithm is followed by the sensitivity measure the feature selection was based on (*e.g.* LinSVM on 50(SVM) reads: linear SVM classifier using 50 features selected by their magnitude of weight from a trained linear SVM).

	Training			**Transfer**	
Classifier	**Features utilized**	**Error**	**Time (sec)**	**Error (±stderr)**	**Time (sec)**
Without feature selection					
LinSVM(C=def)	29125	0.00	22.7	0.40±0.07	2.0
LinSVM(C=10*def)	29125	0.00	22.7	0.36±0.07	2.0
LinSVM(C=1)	29125	0.00	22.6	0.36±0.07	2.0
RbfSVM()	29125	0.00	23.8	0.50±0.07	2.1
kNN()	29125	0.02	0.0	0.44±0.03	2.9
With feature selection					
SMLR(lm=0.1)	314	0.00	19.0	0.09±0.03	0.1
SMLR(lm=1.0)	92	0.00	5.0	0.11±0.03	0.1
SMLR(lm=10.0)	42	0.00	2.4	0.09±0.03	0.1
RbfSVM on SMLR(lm=10) non-0	42	0.00	2.5	0.11±0.02	0.0
kNN on 5%(ANOVA)	1456	0.00	0.8	0.28±0.05	0.2
kNN on 50(ANOVA)	50	0.01	0.8	0.07±0.02	0.0
kNN on SMLR(lm=10) non-0	42	0.00	2.5	0.12±0.02	0.0
LinSVM on 5%(SVM)	1456	0.00	23.1	0.18±0.04	0.1
LinSVM on 50(SVM)	50	0.00	22.7	**0.03**±0.02	0.0
LinSVM on 5%(ANOVA)	1456	0.00	1.6	0.13±0.04	0.1
LinSVM on 50(ANOVA)	50	0.00	0.8	0.09±0.03	0.0
LinSVM on SMLR(lm=1) non-0	92	0.00	5.3	0.08±0.02	0.0
LinSVM on SMLR(lm=10) non-0	42	0.00	2.5	0.12±0.03	0.0
LinSVM+RFE(N-Fold)	4587	0.00	2010.0	0.12±0.03	3.4
LinSVM+RFE(OddEven)	42	0.09	260.9	0.24±0.04	0.0