PyMVPA: a unifying approach to the analysis of neuroscientific data

Michael Hanke1,2†, Yaroslav O. Halchenko3,4,5†, Per B. Sederberg6,7, Emanuele Olivetti8,9, Ingo Fründ1,10,11, Jochem W. Rieger2,12,13, Christoph S. Herrmann1,11,13, James V. Haxby14,15, Stephen José Hanson3,5 and Stefan Pollmann1,2,13*

1 Department of Psychology, University of Magdeburg, Magdeburg, Germany
2 Center for Advanced Imaging, Magdeburg, Germany
3 Psychology Department, Rutgers Newark, New Jersey, USA
4 Computer Science Department, New Jersey Institute of Technology, Newark, New Jersey, USA
5 Rutgers University Mind Brain Analysis, Rutgers Newark, New Jersey, USA
6 Department of Psychology, Princeton University, Princeton, New Jersey, USA
7 Princeton Neuroscience Institute, Princeton University, Princeton, New Jersey, USA
8 Center for Information Technology (Irist), Fondazione Bruno Kessler, Trento, Italy
9 Center for Mind/Brain Sciences (CIMeC/NILab), University of Trento, Italy
10 Leibniz Institute for Neurobiology, Magdeburg, Germany
11 Bernstein Group for Computational Neuroscience, Magdeburg, Germany
12 Department of Psychology, Princeton University, Princeton, New Jersey, USA
13 Center for Behavioral Brain Sciences, Magdeburg, Germany
14 Center for Cognitive Neuroscience, Dartmouth College, Hanover, New Hampshire, USA
15 Department of Psychological and Brain Sciences, Dartmouth College, Hanover, New Hampshire, USA

Edited by: Rolf Kötter, Radboud University Nijmegen, The Netherlands
Reviewed by: Martin A. Spacek, The University of British Columbia, Canada
Samuel Garcia, Université Claude Bernard Lyon I, France

*Correspondence: Stefan Pollmann, Institut für Psychologie II, Otto-von-Guericke-Universität Magdeburg, PF 4120, D-39016 Magdeburg, Germany. e-mail: stefan.pollmann@ovgu.de
†Hanke and Halchenko contributed equally to this article.

Received: 14 September 2008; paper pending published: 20 October 2008; accepted: 20 January 2009; published online: 02 February 2009.

All analyses that were performed for the paper followed the same steps: data input, preprocessing, analysis, and presentation of the results. The analysis stage was nearly identical for all modalities and was carried out by the doSensitivityAnalysis() function shown below. The other steps deviated from a common line due to specifics of each modality data (e.g., storage, shape), and the summary plots to be presented.

The following sections provide example code snippets from the complete source code to illustrate various aspects of the analysis pipeline. Complete sources of the analysis are also available from http://www.pymvpa.org/pubs/frontiers2008. Source code is guaranteed to be compatible with PyMVPA version 0.3.11. For more analysis code examples of PyMVPA, and the explanation of PyMVPA building blocks, we would like to refer the reader to PyMVPA User Manual2.

INPUT

Datasets for the analyses were obtained from different sources and in different storage formats: from NIFTI files of fMRI volumes, to Mathworks Matlab .mat files for extracellular recordings. Consequently, each required modality-specific handling of input data, which was carried out in loadData() methods of the scripts. The following example code to load the dataset is from fMRI analysis code:

```python
1 def loadData(subj):
2     """Load data for one subject and return
dataset.
3     :Parameter:
4     subj: str
5     ID of the subject who's data should be
6     loaded.
7     :Returns:
8     NiftiDataset instance.
9     """
10    verbose(1, "Loading fMRI data from basepath 
11       %s % datapath)" % datapath)
12    # load labels and chunk information from
13    # file layout: one line per voxel, two rows
14    # (label(str), chunk(int)) chunk corresponds
15    # to the experimental run where the volume
```

1See http://www.pymvpa.org/#download for download and installation instructions.
2http://www.pymvpa.org/contents.html.
The PyMVPA framework

PREPROCESSING

The data preprocessing stage prior to applying machine learning is highly modality-dependent, but PyMVPA provides a number of preprocessing mechanisms. For EEG/MEG it is typical first to filter and down-sample the data, which can be done with resample method of a Dataset:

```python
def resample(d, sr):
    if sr != d.samplingrate:
        d.resample(sr=sr)
```

Optionally, data per each trial could be transformed using wavelet discrete or packet transformations:

```python
if options.wavelet_decomposition == 'dwt':
    WT = WaveletTransformationMapper(**kwargs)
else:
    WT = WaveletPacketMapper(**kwargs)
```

Many machine learning methods require data to be in a suitable range of values, typically to lie mostly within $[-1,1]$ range. In PyMVPA we typically transform data per each feature into $z$-scores, optionally relative to some baseline condition:

```python
zscore(ds, perchunk=True, targetdtype='float32')
```

ANALYSIS

The machine-learning analysis for all modalities was performed with the same module `warehouse.py`, which imports the full PyMVPA suite and defines the function modality-independent (doSensitivityAnalysis):

```python
def doSensitivityAnalysis(ds, clfs, sensanas, splitter, sa_args=' '):
```

Parameters:
- `ds`: Dataset
- `clfs`: list of Classifier
- `sensanas`: list of DatasetMeasure
- `splitter`: Splitter
- `sa_args`: basestring

"""Generic function to perform sensitivity analysis (along classification"

:Parameters:
- `ds`: Dataset to perform analysis on
- `clfs`: list of Classifier
- `sensanas`: list of DatasetMeasure
- `splitter`: Splitter
- `sa_args`: basestring

Additional optional arguments to provide to getSensitivityAnalyzer

optional arguments to provide to getSensitivityAnalyzer

"""
# to absorb all sensitivities
senses = []

# run classifiers in cross-validation
for label, clf in clfs.iteritems():
    sclf = SplitClassifier(clf, splitter,
        enable_states=[' confusion ',
            ' training_confusion '])

    verbose(1, ' Doing cross-validation with ' + label)

    # Compute sensitivity, which in turn
    # trains the sclf
    sensitivities = \
        sclf.getSensitivityAnalyzer(
            combiner=None, slave_combiner=None)(ds)

    verbose(1, ' Accumulated confusion matrix ' + str(sclf.confusion))

    # and store
    senses.append(
        (label + ' ( %.1f %% corr.) weights ' %
            sclf.confusion.stats[' ACC %'],
            sensitivities, sclf.confusion,
            sclf.training_confusion))

# Simple 'print cv.confusion' provides the same output
# without the description of abbreviations
print cv.confusion.asstring(description=True)

<table>
<thead>
<tr>
<th></th>
<th>predict.</th>
<th>3kHz</th>
<th>7kHz</th>
<th>12kHz</th>
<th>20kHz</th>
<th>30kHz</th>
<th>song1</th>
<th>song2</th>
<th>song3</th>
<th>song4</th>
<th>song5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>17 4 17 4 16 1 0 0 0 150 19 198 35 173</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>1560 1560 1561 1561 1561 1561 1561 1561 1561 1561 1561 1561</td>
<td>84 94 103 158 162 160 171 170 139 136 125 121</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TP</td>
<td>17 4 17 4 16 1 0 0 0 150 19 198 35 173</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TN</td>
<td>1261 1251 1242 1187 1183 1185 1174 1175 1206 1241</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary:

<table>
<thead>
<tr>
<th></th>
<th>predict.</th>
<th>3kHz</th>
<th>7kHz</th>
<th>12kHz</th>
<th>20kHz</th>
<th>30kHz</th>
<th>song1</th>
<th>song2</th>
<th>song3</th>
<th>song4</th>
<th>song5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>17 4 17 4 16 1 0 0 0 150 19 198 35 173</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>1560 1560 1561 1561 1561 1561 1561 1561 1561 1561 1561 1561</td>
<td>84 94 103 158 162 160 171 170 139 136 125 121</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TP</td>
<td>17 4 17 4 16 1 0 0 0 150 19 198 35 173</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TN</td>
<td>1261 1251 1242 1187 1183 1185 1174 1175 1206 1241</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Means:

<table>
<thead>
<tr>
<th></th>
<th>predict.</th>
<th>3kHz</th>
<th>7kHz</th>
<th>12kHz</th>
<th>20kHz</th>
<th>30kHz</th>
<th>song1</th>
<th>song2</th>
<th>song3</th>
<th>song4</th>
<th>song5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>17 4 17 4 16 1 0 0 0 150 19 198 35 173</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>1560 1560 1561 1561 1561 1561 1561 1561 1561 1561 1561 1561</td>
<td>84 94 103 158 162 160 171 170 139 136 125 121</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TP</td>
<td>17 4 17 4 16 1 0 0 0 150 19 198 35 173</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TN</td>
<td>1261 1251 1242 1187 1183 1185 1174 1175 1206 1241</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RESULTS

Given that the research goals and original data structures differ across modalities, the presentation of the results was also performed for each modality separately. Some aspects as generalization performance presentation, however, is common to any application of the classifiers.
GENERALIZATION PERFORMANCE

PyMVPA is equipped with easy ways to have a glance overview over the generalization performance of a cross-validated classifier. Such summary is provided by a `ConfusionMatrix` class, and is accompanied by various performance metrics. For example, the 8-fold cross-validation of the extracellular recordings with the SMLR classifier produced the above confusion matrix on page 3:

Statistics computed in 1-vs-rest fashion per each target.

Abbreviations (for details see http://en.wikipedia.org/wiki/ROC_curve):
- TP : true positive (AKA hit)
- TN : true negative (AKA correct rejection)
- FP : false positive (AKA false alarm, Type I error)
- FN : false negative (AKA miss, Type II error)
- TPR: true positive rate (AKA hit rate, recall, sensitivity)
  TPR = TP / (TP + FN)
- FPR: false positive rate (AKA false alarm rate, fall-out)
  FPR = FP / (FP + TN)
- ACC: accuracy
  ACC = (TP + TN) / (P + N)
- SPC: specificity
  SPC = TN / (FP + TN) = 1 − FPR
- PPV: positive predictive value (AKA precision)
  PPV = TP / (TP + FP)
- NPV: negative predictive value
  NPV = TN / (TN + FN)
- FDR: false discovery rate
  FDR = FP / (FP + TP)
- MCC: Matthews Correlation Coefficient
  MCC = (TP*TN − FP*FN)/sqrt(P N P' N')

# of sets: number of target/prediction sets which were provided

In addition to the abusively informative textual representation, there is an alternative graphical representation of the confusion matrix (Figure 1) via the `.plot()` method of a `ConfusionMatrix` instance (e.g., `cv.confusion.plot(); P.show()`).

SENSITIVITIES PLOTS

PyMVPA was not designed to be a visualization toolkit, thus it has only a few helpers to plot common types of figures (e.g., `plotERP`, for event-related plots). Any custom 2D-plot can easily be crafted with `Matplotlib`. In addition, some modules (e.g., `hcluster`) provide specific plotting functions that accompany their packages.

The following code snippet lists two functions that produced the results in the "EEG Analysis" section of the article:

```python
def finalFigure(ds_pristine, ds, senses, channel):
    """Generate final ERP, sensitivity and topography plots"""
    # Parameters:
    ds_pristine: Dataset
    ds: Dataset
    Dataset as used for the sensitivity analyses to generate sensitivity and topography plots
    senses: list of 2-tuples
    (sensit. ID, sensitivities (nfolds x nfeatures)
    The sensitivities used to select a subset of voxels in each ROI
    channel: str
    Id of the channel to be used for ERP and sensitivity plots over time.
    
    # sampling rate
    SR = ds_pristine.samplingrate
    # data is already trials, this would correspond sec before onset
    pre = -(int(ds_pristine.t0*100)/100.0)
    pre = round to 2 digits
    # number of channels, samples per trial
    nchannels, spt = \
        ds_pristine.mapper.mask.shape
    post = spt * 1.0/ SR − pre
    # compute seconds in trials after onset
    ch_of_interest = \
        ds_pristine.channelids.index(channel)
```

FIGURE 1 | Multi-unit extracellular recordings: confusion matrix among 10 stimuli of 2 groups: pure tones and natural sounds.

http://code.google.com/p/scipy-cluster/.

# error type to use in all plots
errtype=[' std ', ' ci95 ']

fig = P.figure(facecolor=' white ',
               figsize=(12, 6))

# plot ERPs
ax = fig.add_subplot(2, 1, 1,
                      frame_on=False)

# map dataset samples back into original
# (electrode) space
responses = \
    { ' label ': responses[0].mean(axis = 0),
      ' data ': responses[1].mean(axis = 0),
      ndmin = 2 }

# plot them all at once
plotERPs([ {' label ': ' lineart ', ' color ': ' r ', ' data ': responses[0]},
          {' label ': ' picture ', ' color ': ' b ', ' data ': responses[1]},
          {' label ': ' dwave ', ' color ': ' 0 ', ' data ': dwave, ' pre_mean ': 0}],
         pre=pre, pre_mean=pre,
         post=post, SR=SR, ax=ax, errtype=' ci95 ',
         ylabel=None, ylformat=' %.2f ',
         pre_mean=0)

# add a legend to the figure
P.legend(sens_labels)

return fig

def topoFigure(ds, senses):
    
    # how many sensitivities do we have
    nsens = len(senses)

    # new figure for topographies
    fig = P.figure(facecolor=' white ',
                   figsize=((nsens+1)*3, 4))

    # again for all available sensitivities
    for i, sens_ in enumerate(senses):
        (sens_id, sens) = sens_[:2]
        sens_labels.append(sens_id)

        # back-project into electrode space
        backproj = ds.mapReverse(sens)

        # and normalize so that all non-zero
        # weights sum up to 1
        # ATTN: need to norm sensitivities for
        # each fold on their own -- who knows
        # what's happening otherwise
        for f in xrange(backproj.shape[0]):
            backproj[f] = L2Normed(backproj[f])

        # take one channel: yields
        ch_sens = backproj[:, 0]

        if ch_sens.mean() < 0:
            ch_sens *= -1

        # just ci95 error here, due to the low
        # number of folds not much different
        # from std; also do _not_ demean based on
        # initial baseline as we want the
        # untransformed sensitivities
        plotERPs(erp_cfgs, pre=pre, post=post,
                  SR=SR, ax=ax, errtype=' ci95 ',
                  ylabel=None, ylformat=' %.2f ',
                  pre_mean=0)

    return fig

# (nfolds x ntimepoints)
ch_sens = backproj[:, ch_of_interest, :]
scores = np.sum(Absolute(averageproj), axis=1)

# strip EOG scores (which are zero anyway, as they had been stripped of before cross-validation)
scores = scores[:-3]

# and normalize so that all scores squared sum up to 1
scores = L2Normed(scores)

# plot all EEG sensor scores
plotHeadTopography(
    scores, sensors.locations(),
    plotsensors=True, resolution=50,
    interpolation='nearest')

# ensure uniform scaling
P.clim(vmin=0, vmax=0.4)

# No need for full title
P.title(re.sub(' .*', '', sens_id))

# just plot name
# to preserve original size

axis = P.axis()

# Draw a color 'bar' for the given sensitivity
ax.bar(-0.4, 0.1, 0.8, 1.4,
       color=colors[i],
       edgecolor=colors[i]);

P.axis(axis)

ax = fig.add_subplot(1, nsens+1, nsens+1,
                     frame_on=False)

cb = P.colorbar(shrink=0.95, fraction=0.05,
                drawedges=False,
                ticks=[0, 0.1, 0.2, 0.3, 0.4])

ax.axison = False

fig.subplots_adjust(left=0.06, right=1.05,
                    bottom=0.01,
                    wspace=-0.2)

P.show()

return fig