Homework 8

Part One now

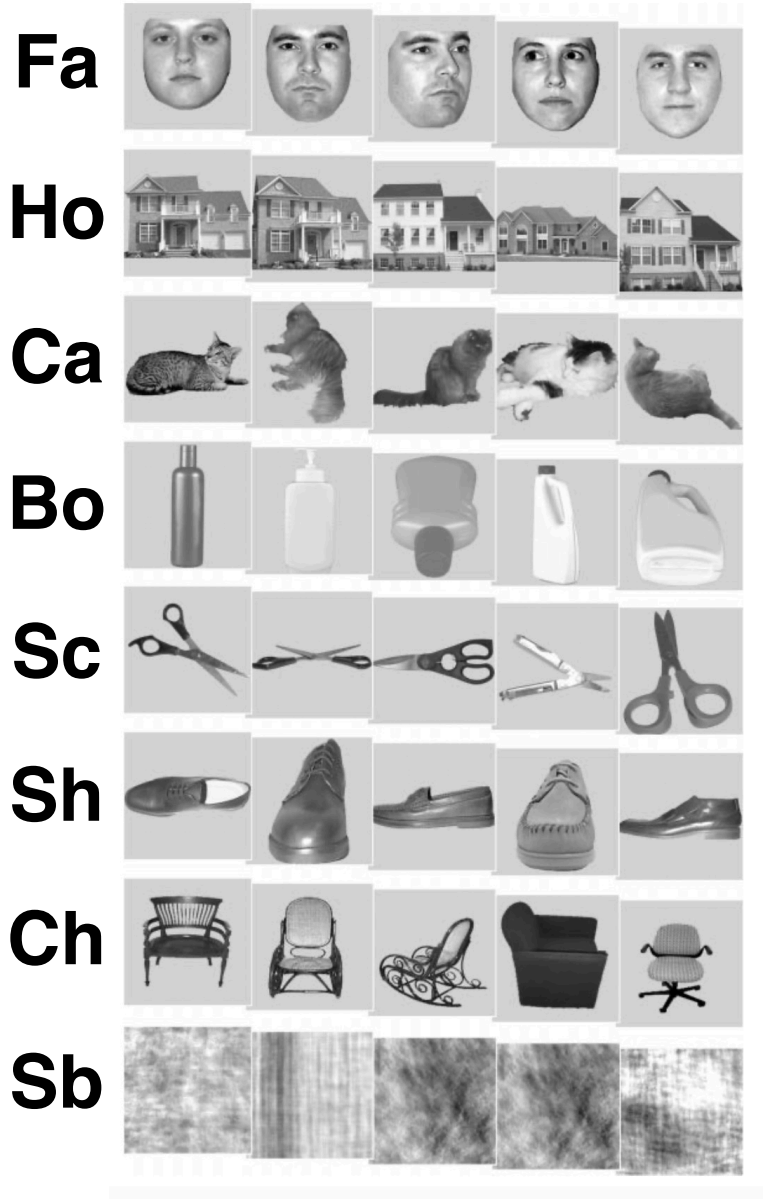
Part Two coming soon

due last day of class (Thu Dec 8)

Part One : reanalysis of Haxby et al. 2001 data

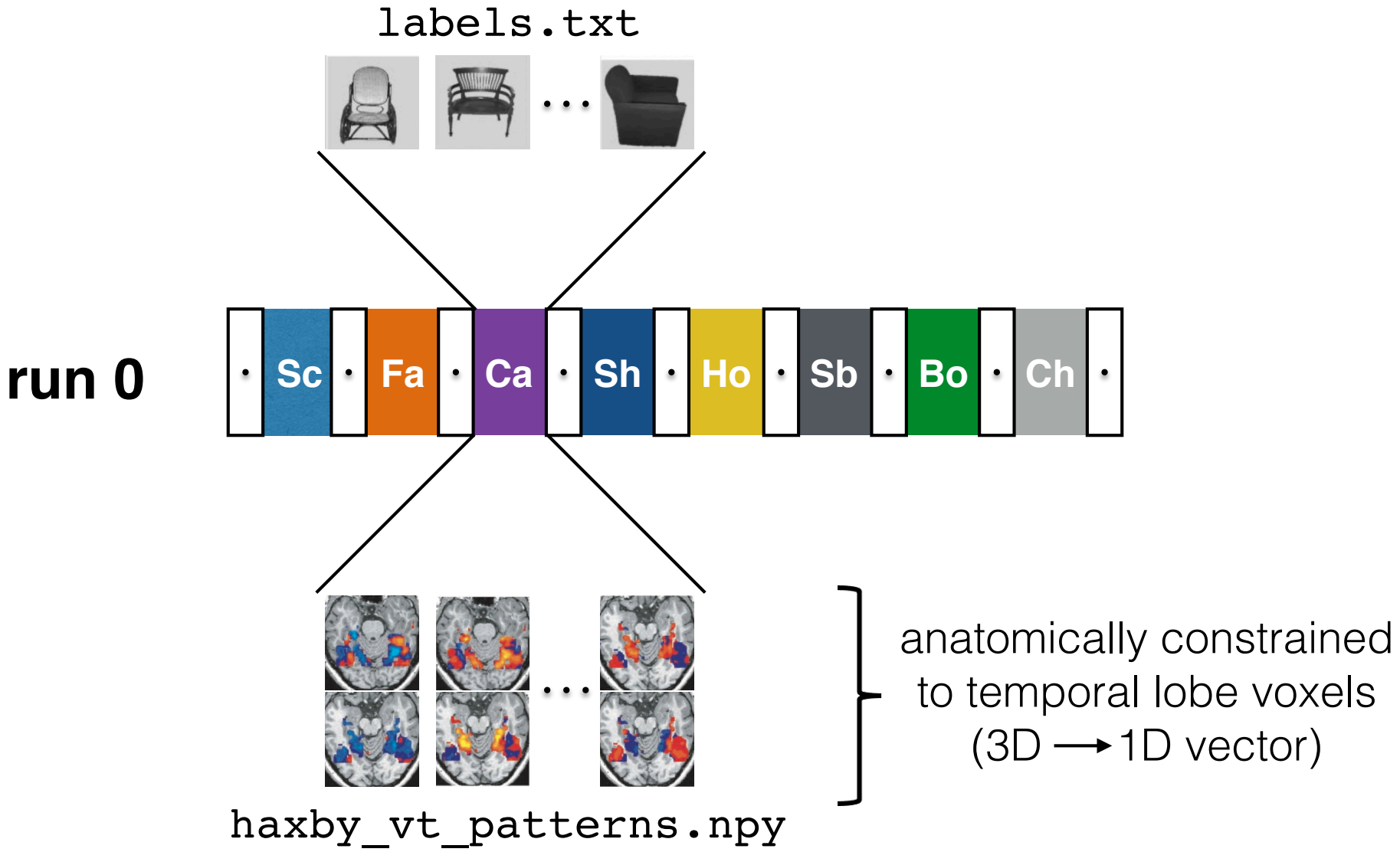
Their original analysis used a simple correlation-based classifier. We will use a simple neural network classifier, trained with backprop.

Haxby et al. 2001 details

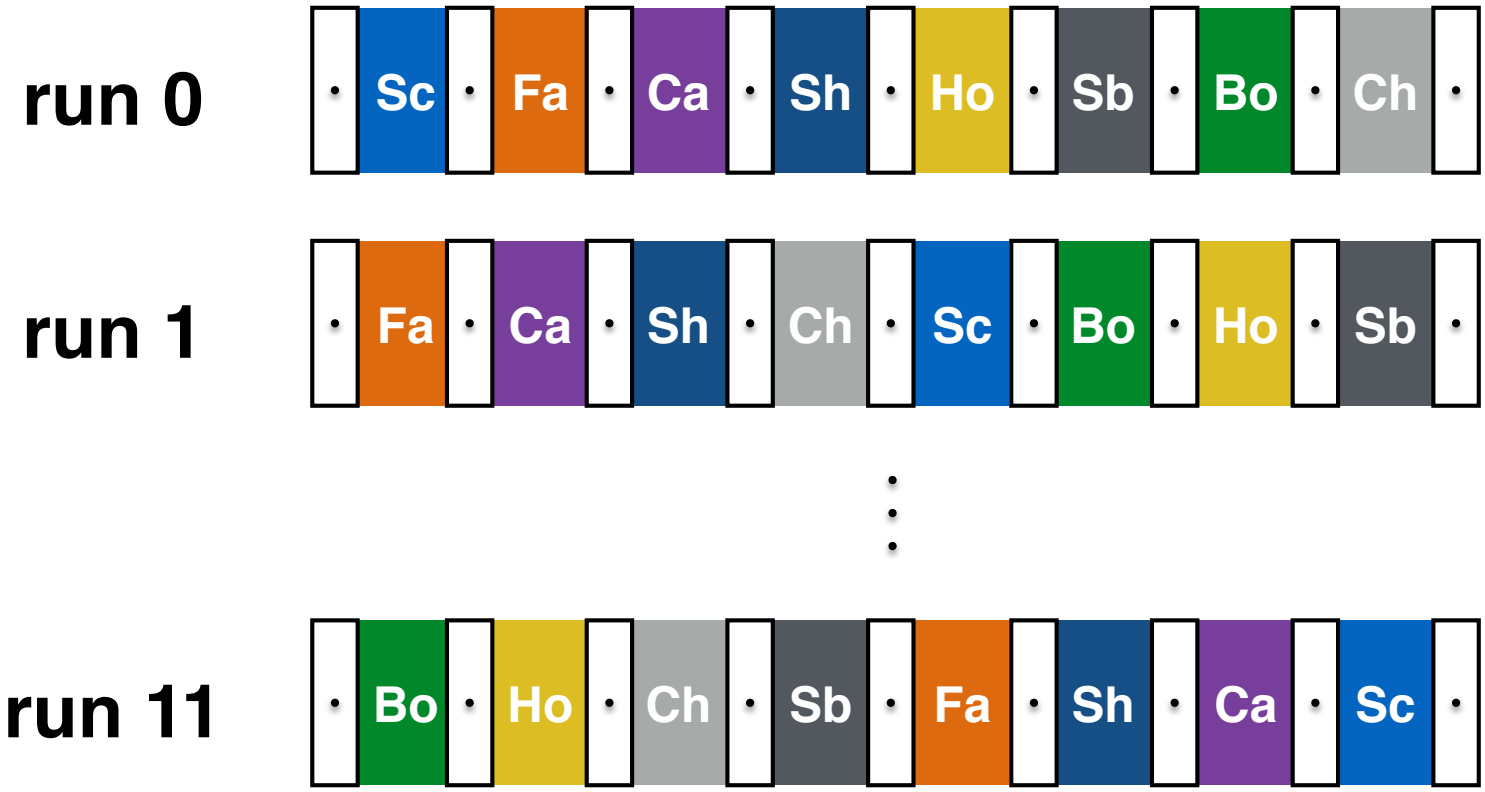


Haxby, J.V., Gobbini, M.I., Furey, M.L., Ishai, A., Schouten, J.L., & Pietrini, P. (2001). Distributed and overlapping representations of faces and objects in ventral temporal cortex. *Science*, 293, 2425-2430.

Haxby et al. 2001 details



Haxby et al. 2001 details



Haxby et al. 2001 details

`labels.txt` (on Brightspace)

```
labels run
rest 0
rest 0
rest 0
rest 0
rest 0
rest 0
scissors 0
scissors 0
scissors 0
scissors 0
scissors 0
scissors 0
scissors 0
scissors 0
scissors 0
rest 0
rest 0
rest 0
rest 0
rest 0
rest 0
face 0
face 0
face 0
face 0
face 0
face 0
.
.
.
scissors 11
scissors 11
scissors 11
scissors 11
rest 11
rest 11
rest 11
rest 11
rest 11
rest 11
```

Homework8.ipynb

```
patterns, targets, runs = prep_haxby_data()
```

`prep_haxby_data()`

- opens `haxby_vt_patterns.npy`
- read in `labels.txt`

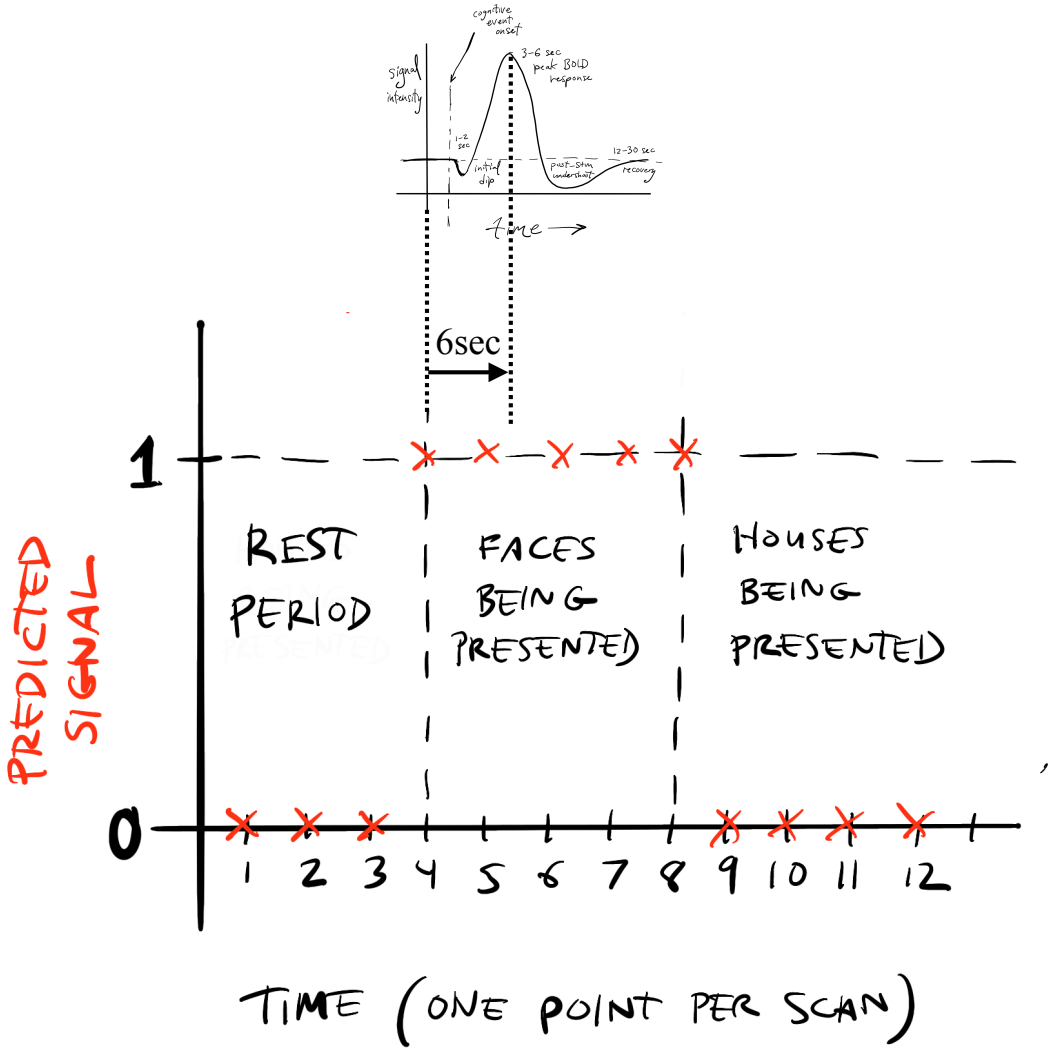
Homework8.ipynb

```
patterns, targets, runs = prep_haxby_data()
```

`prep_haxby_data()`

- opens `haxby_vt_patterns.npy`
- read in `labels.txt`
- "cheap version" of accounting for hemodynamic lag

one approach is to shift predictor function by ~6 sec



Homework8.ipynb

```
patterns, targets, runs = prep_haxby_data()
```

`prep_haxby_data()`

- opens `haxby_vt_patterns.npy`
- read in `labels.txt`
- "cheap version" of accounting for hemodynamic lag

  each scan was 2.5 sec
  shift labels by 5 sec (close to 6 sec)
  which means, shift labels forward by 2 indices

Homework8.ipynb

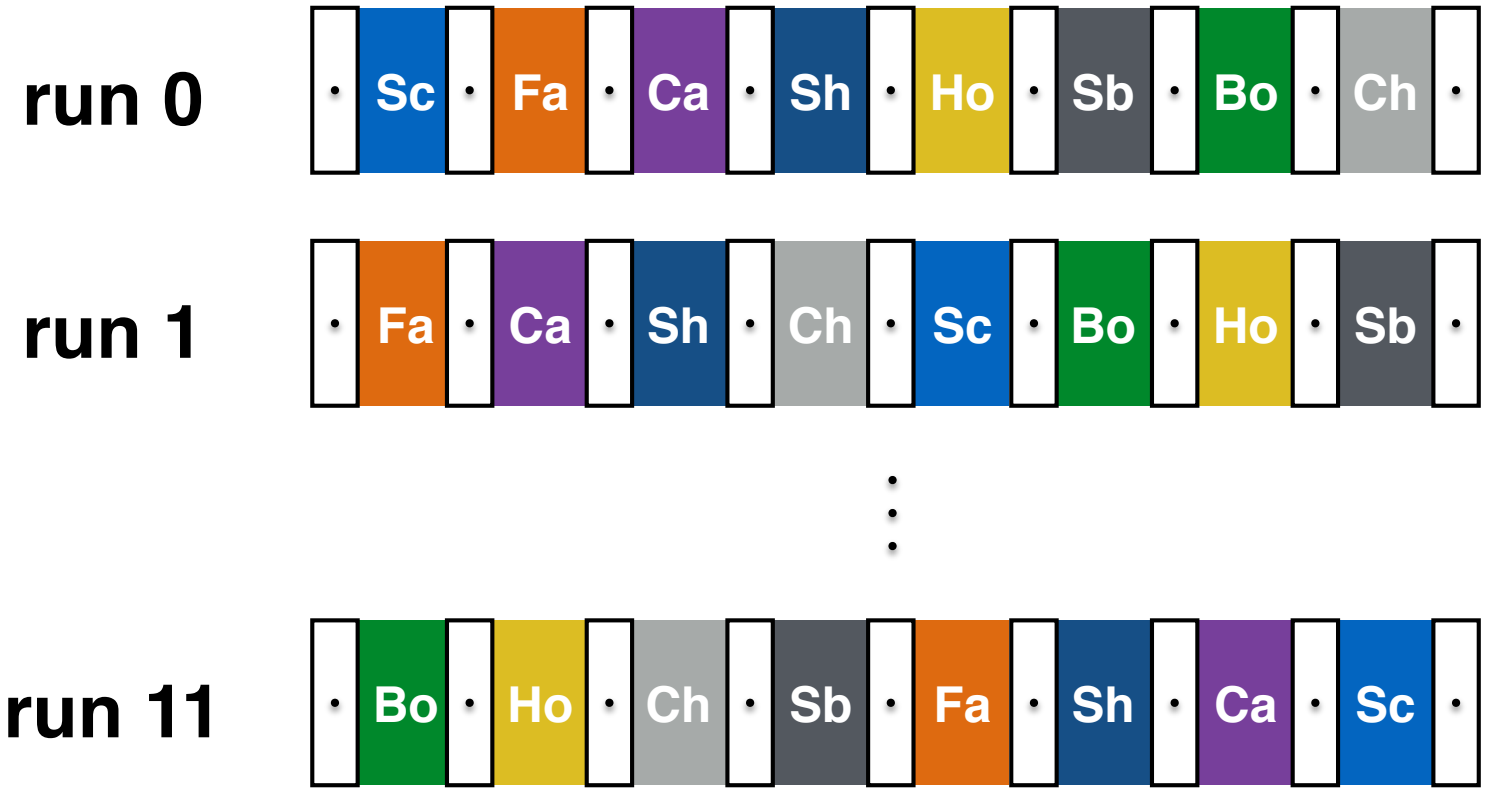
```
patterns, targets, runs = prep_haxby_data()
```

`prep_haxby_data()`

- opens `haxby_vt_patterns.npy`
- read in `labels.txt`
- "cheap version" of accounting for hemodynamic lag
- remove rests from runs

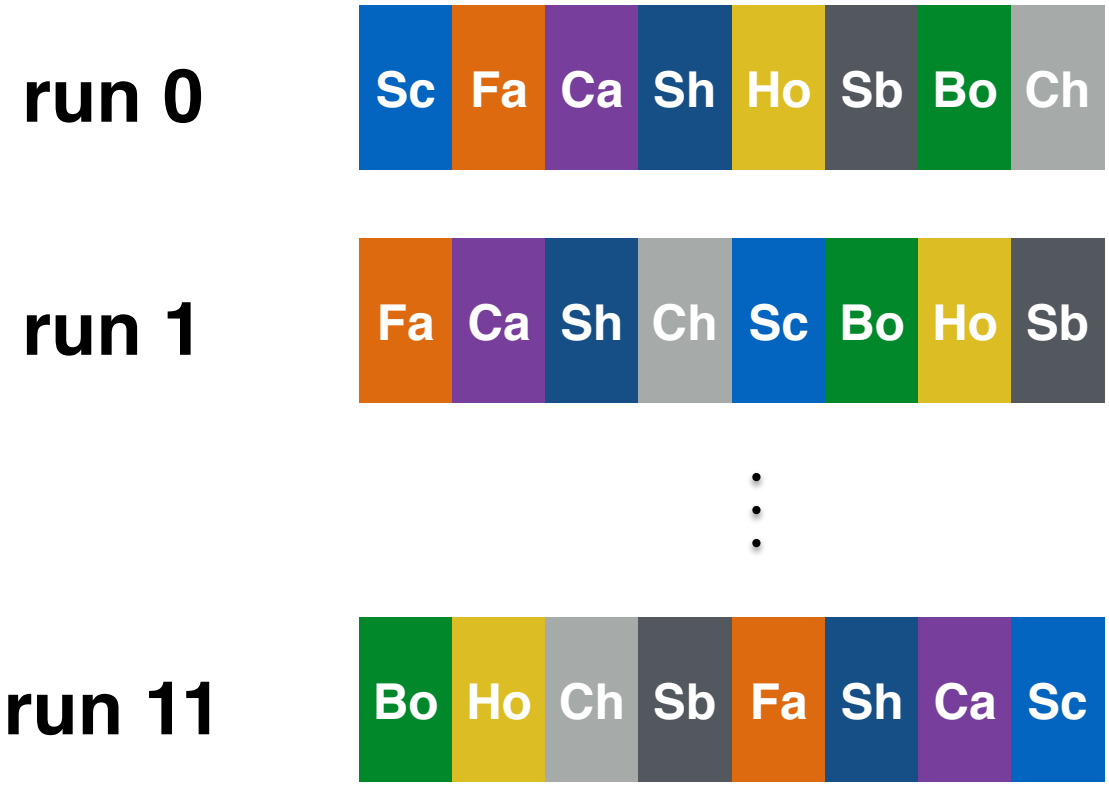
Haxby et al. 2001 details

raw data with rests



Haxby et al. 2001 details

with rests stripped away *from both labels and fMRI data*



Homework8.ipynb

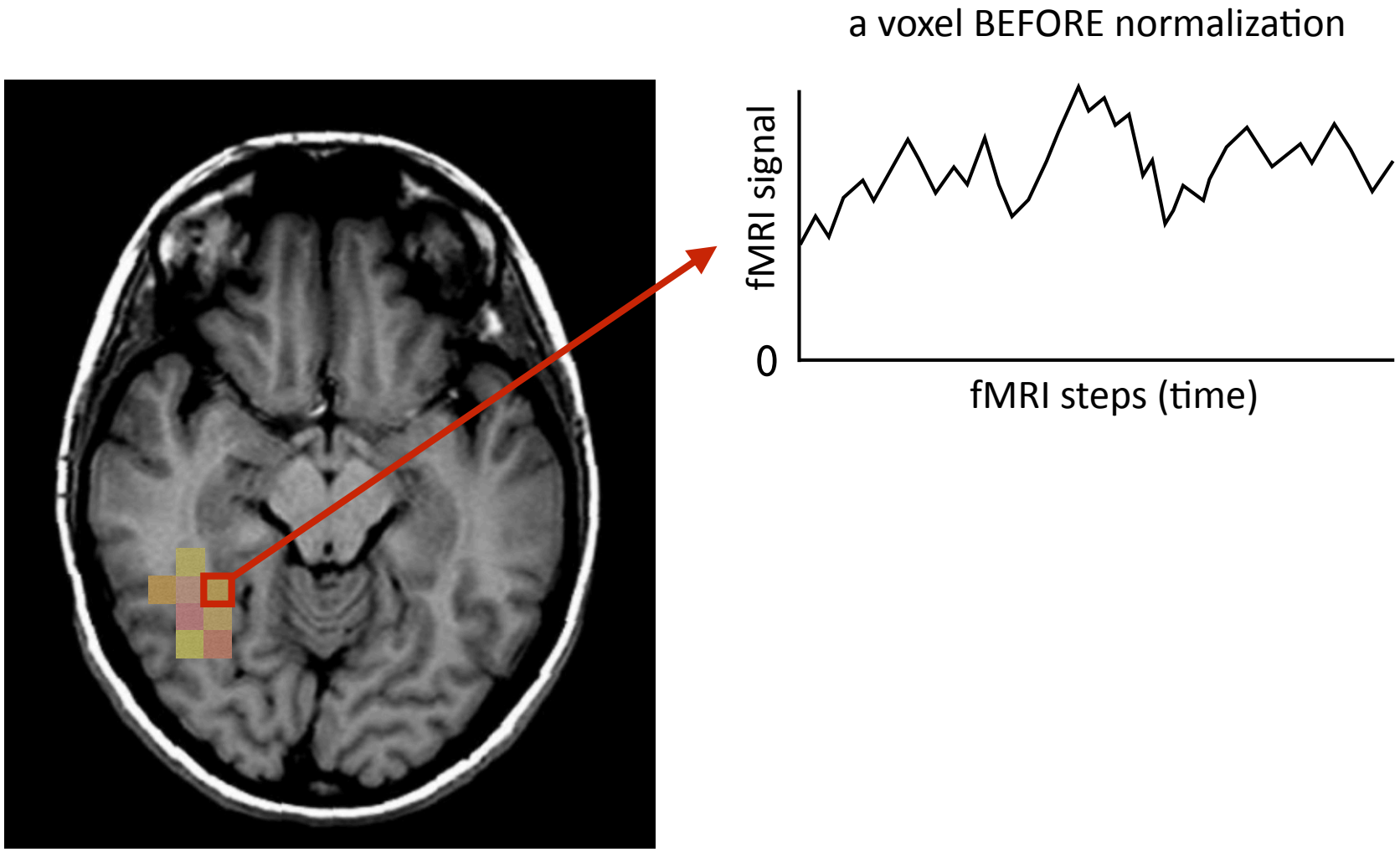
```
patterns, targets, runs = prep_haxby_data()
```

`prep_haxby_data()`

- opens `haxby_vt_patterns.npy`
- read in `labels.txt`
- "cheap version" of accounting for hemodynamic lag
- remove rests from runs
- normalize (z-transform) each voxel

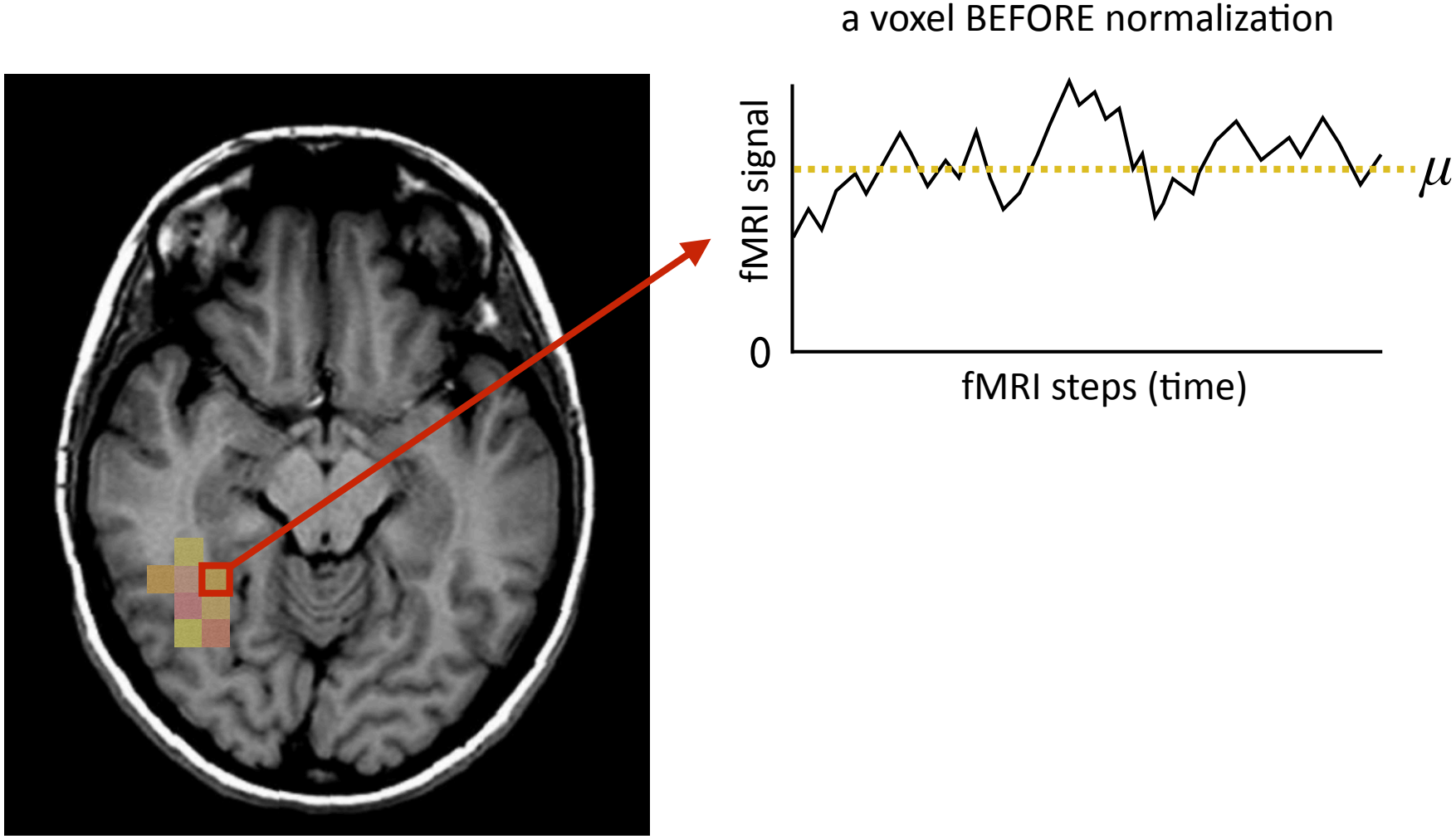
normalize / z-transform

(we've seen before)



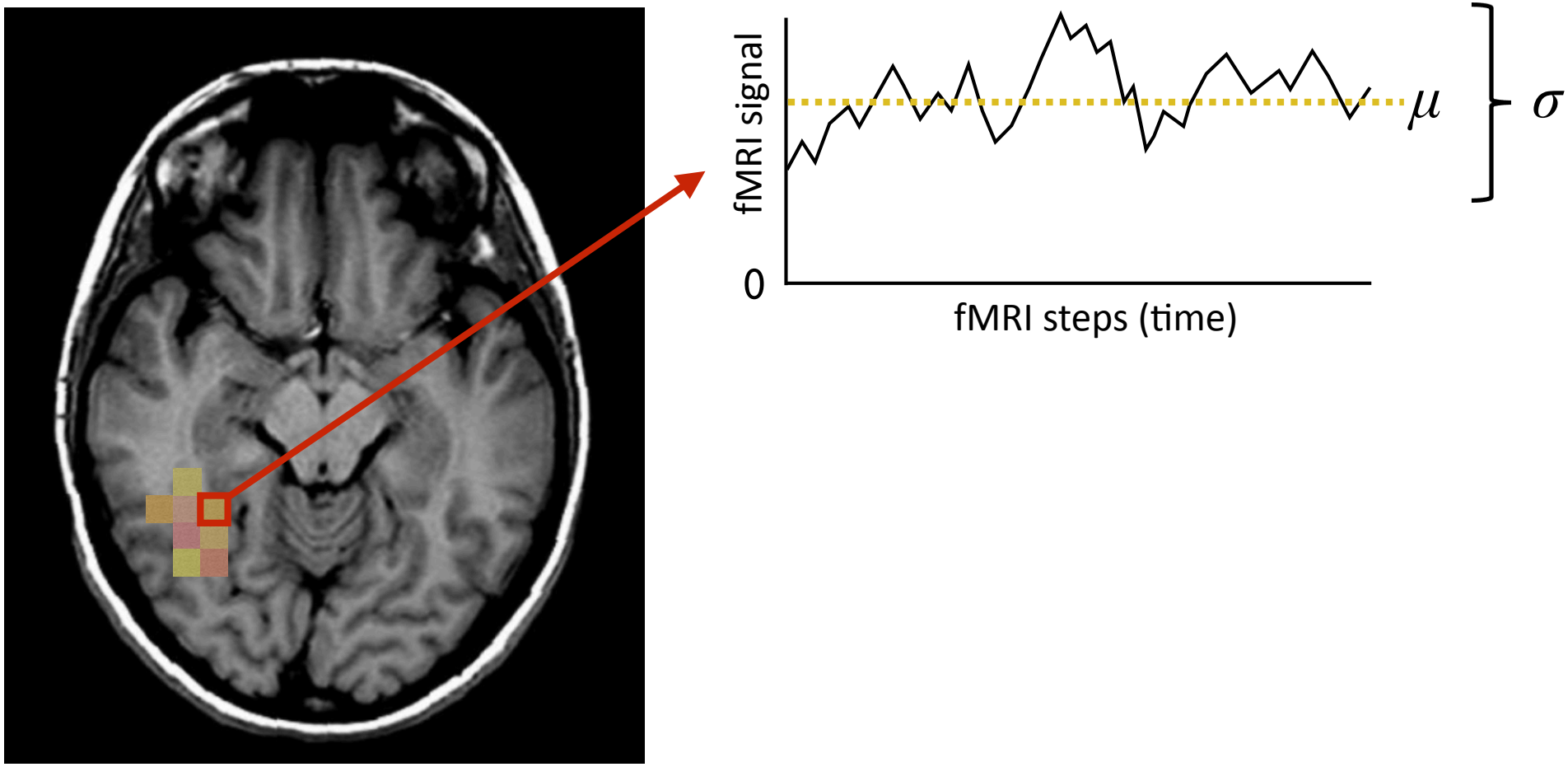
normalize / z-transform

(we've seen before)



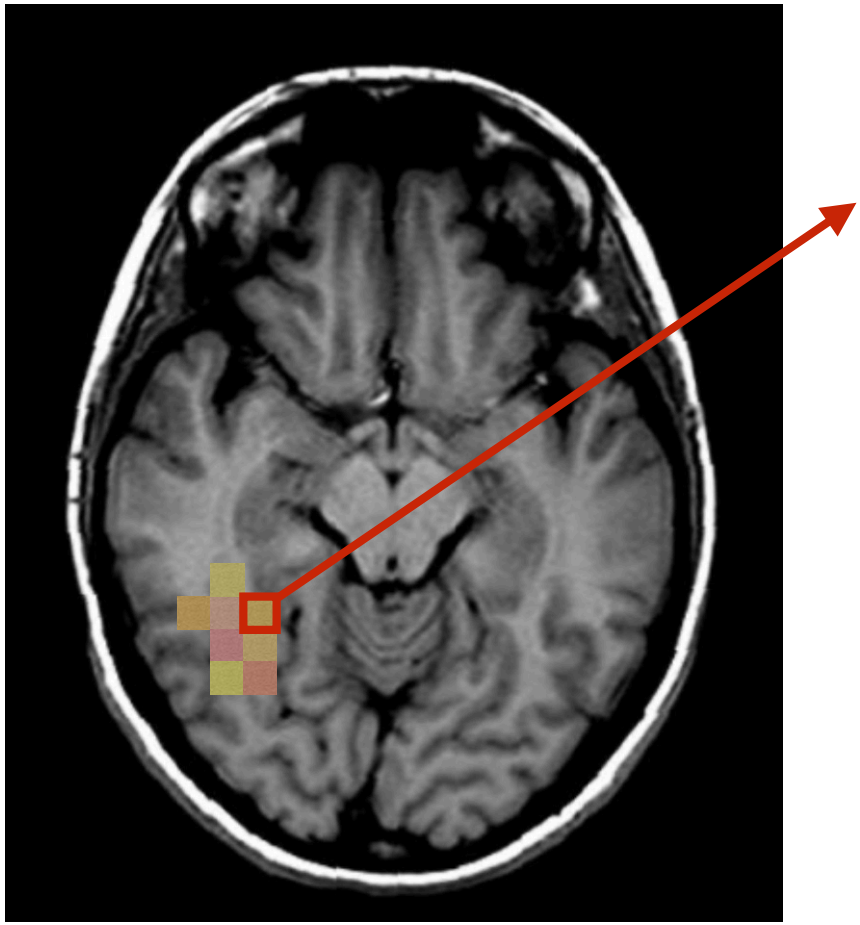
normalize / z-transform

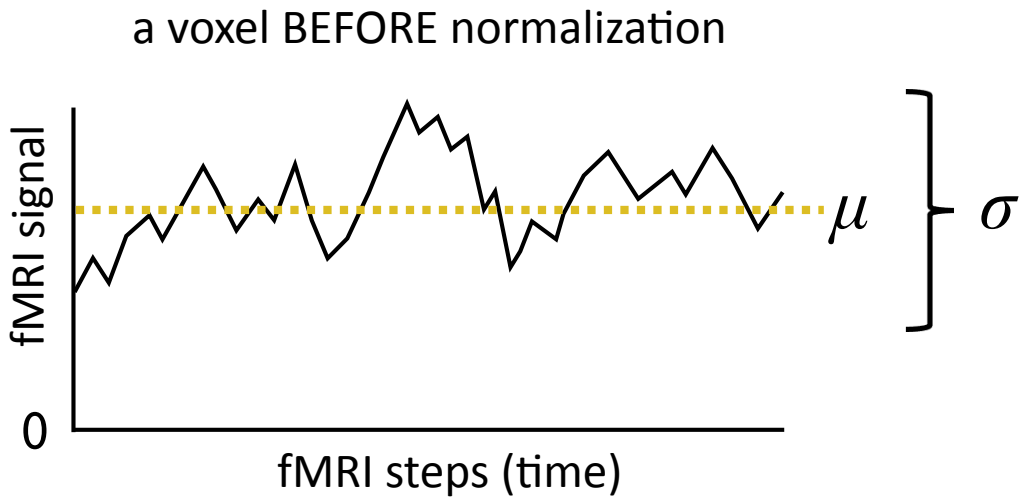
(we've seen before)



normalize / z-transform

(we've seen before)

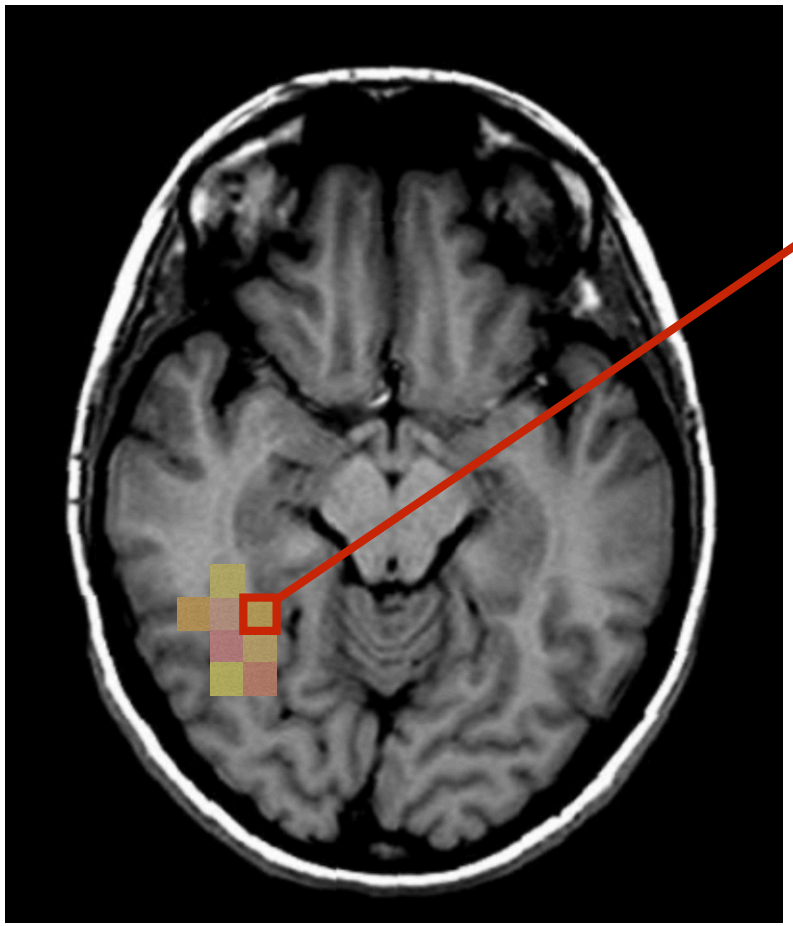




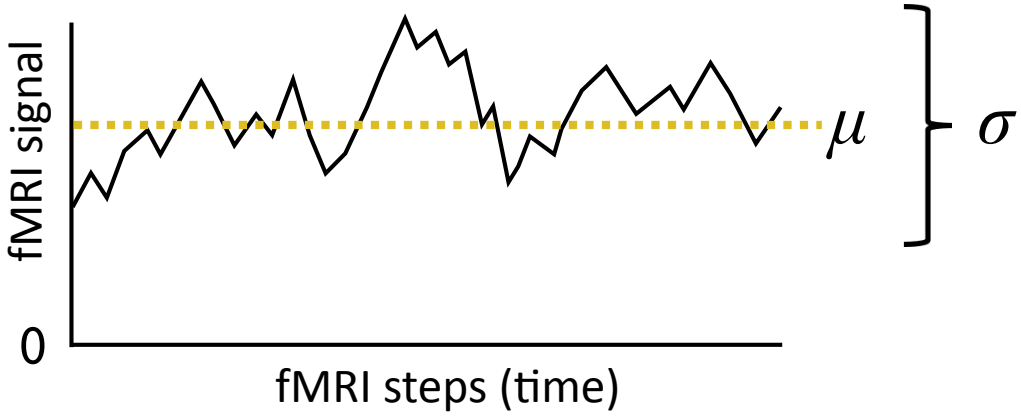
$$x_t = \frac{x_t - \mu}{\sigma}$$

normalize / z-transform

(we've seen before)

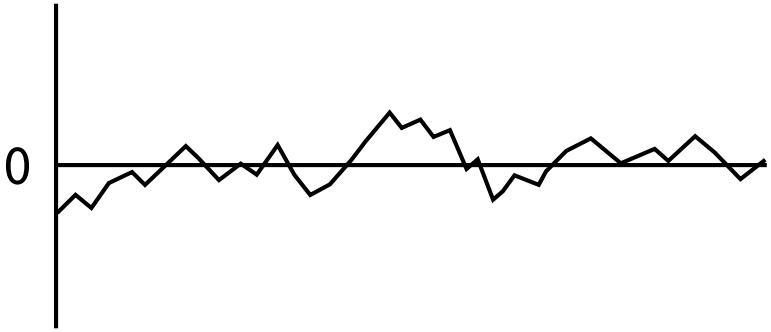


a voxel BEFORE normalization



$$x_t = \frac{x_t - \mu}{\sigma}$$

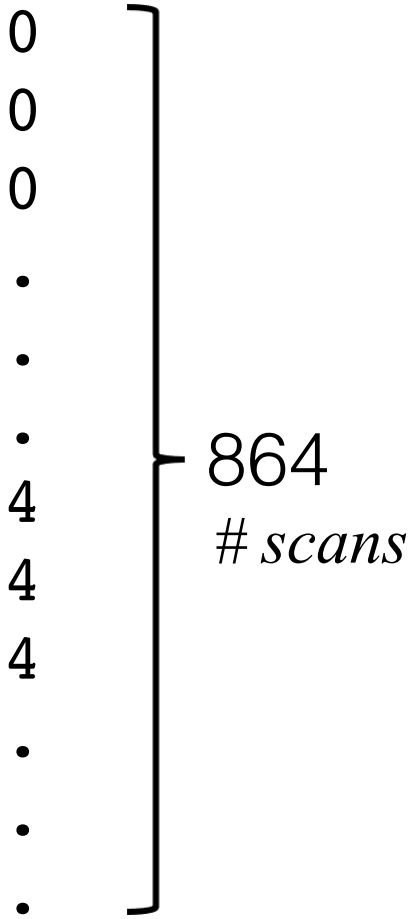
a voxel AFTER normalization



```
patterns, targets, runs = prep_haxby_data()
```

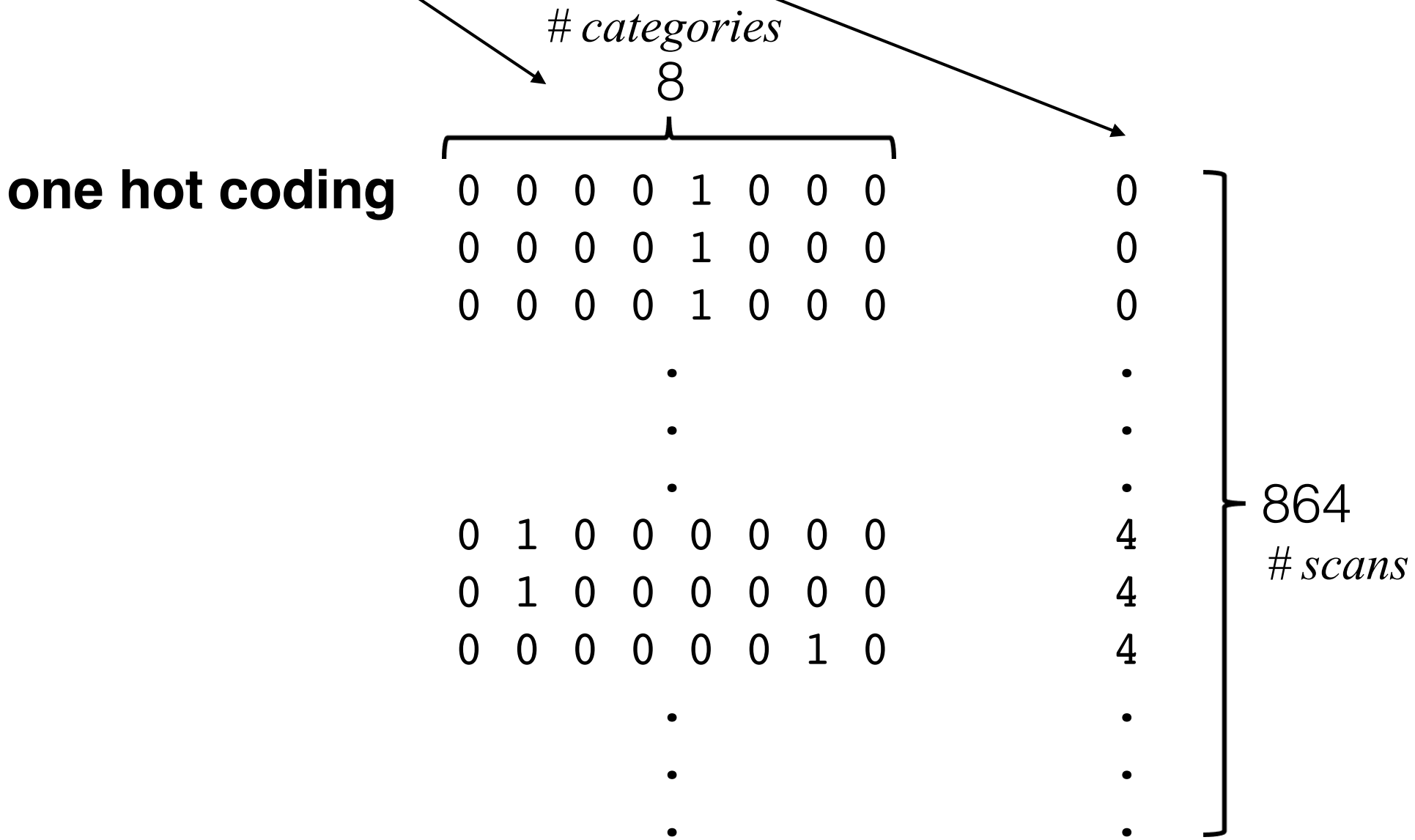
Homework8.ipynb

```
patterns, targets, runs = prep_haxby_data()
```



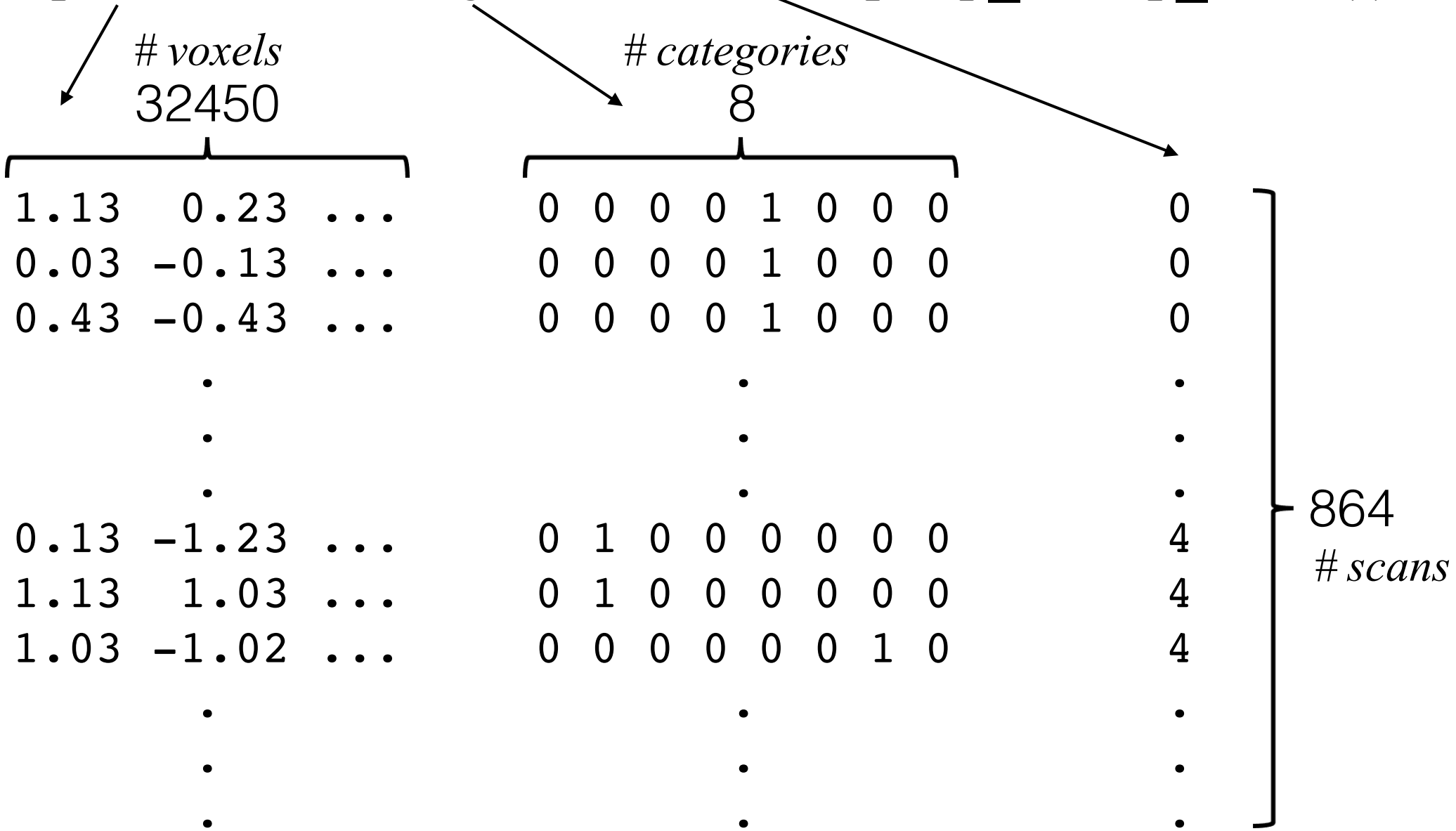
Homework8.ipynb

```
patterns, targets, runs = prep_haxby_data()
```



Homework8.ipynb

```
patterns, targets, runs = prep_haxby_data()
```



"feature selection"

we've narrowed things down to voxels in IT cortex (anatomically)

let's narrow to voxels that show some statistically significant modulation with category (eliminating lower-level and non-visual)

```
def feature_selection...
.
.
.
    sps.f_oneway(groups[0], groups[1], groups[2], groups[3],
                 groups[4], groups[5], groups[6], groups[7])
```

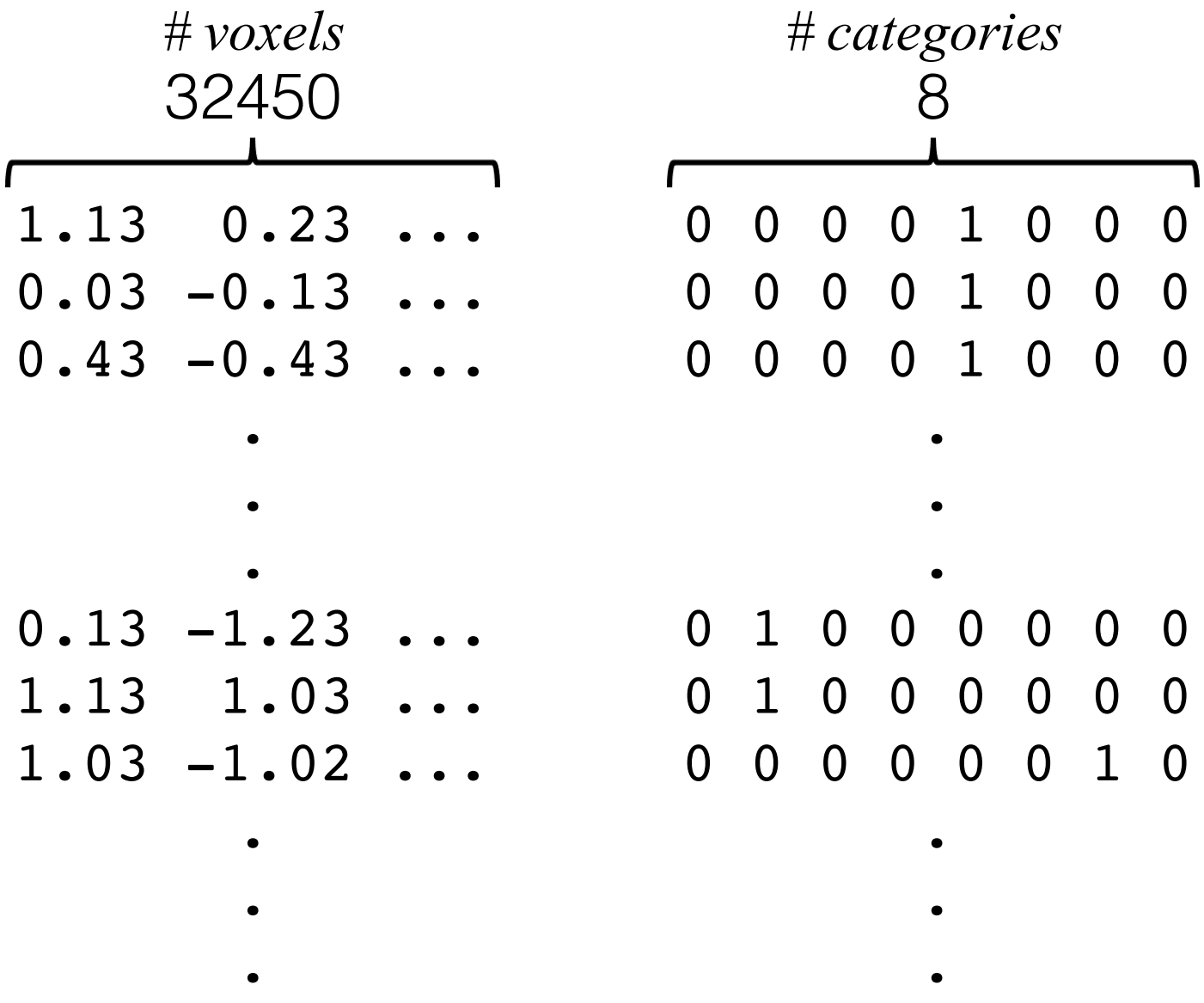
simple one-way ANOVA

# voxels		*# voxels*
32450	→	12432

we'll see where we do this later

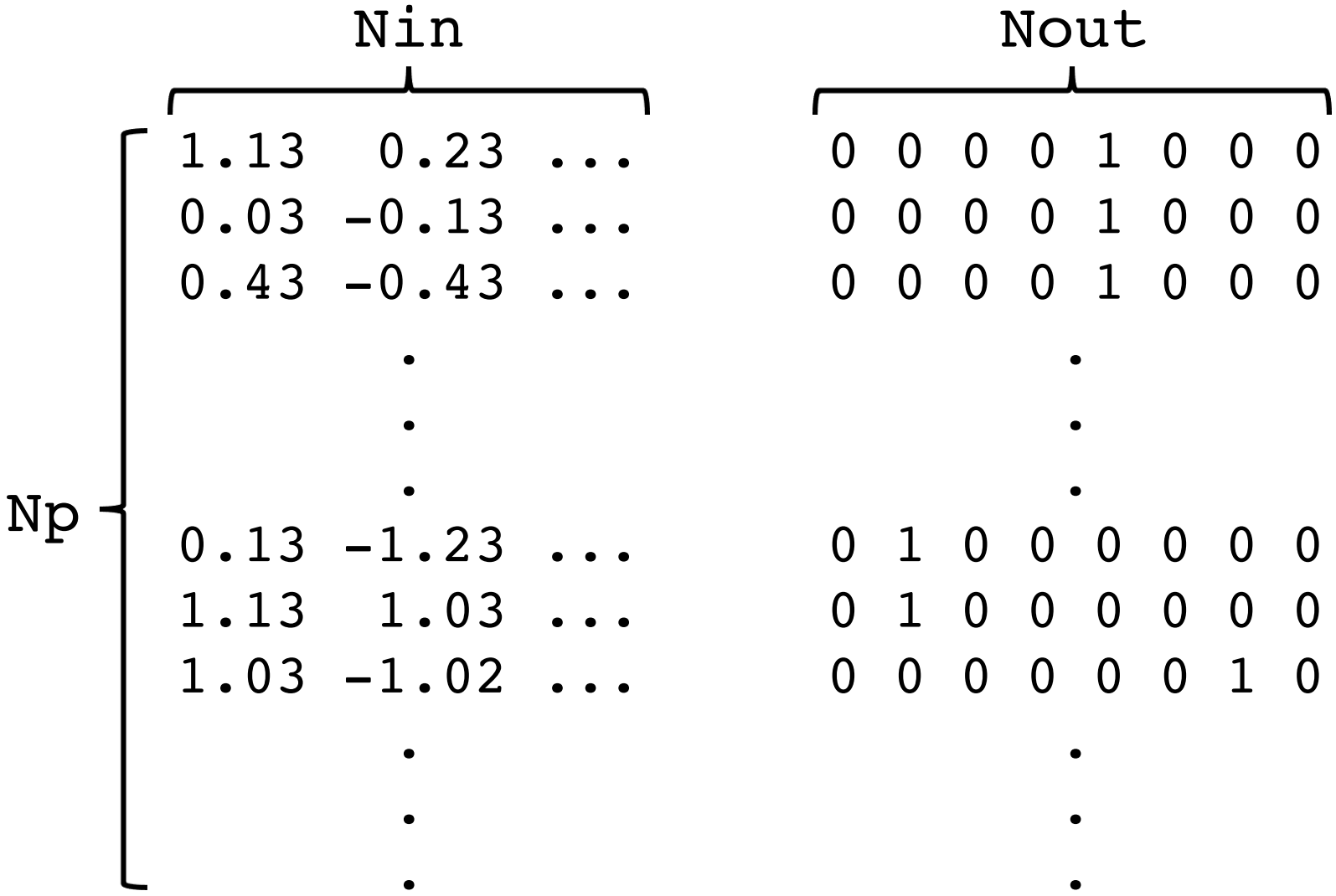
classification in MVPA

it looks like we have enough to train a classifier

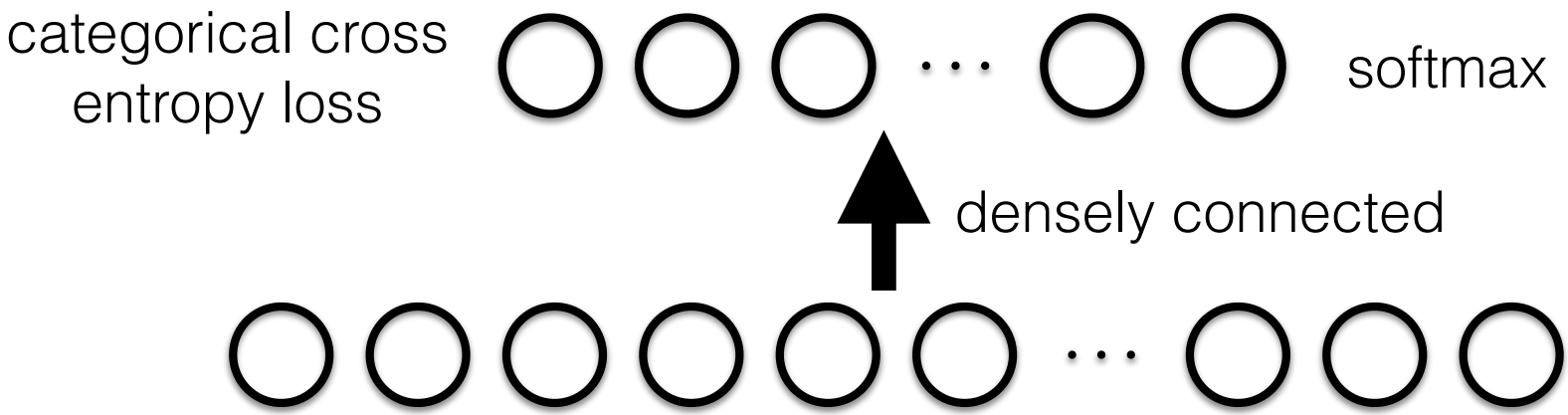


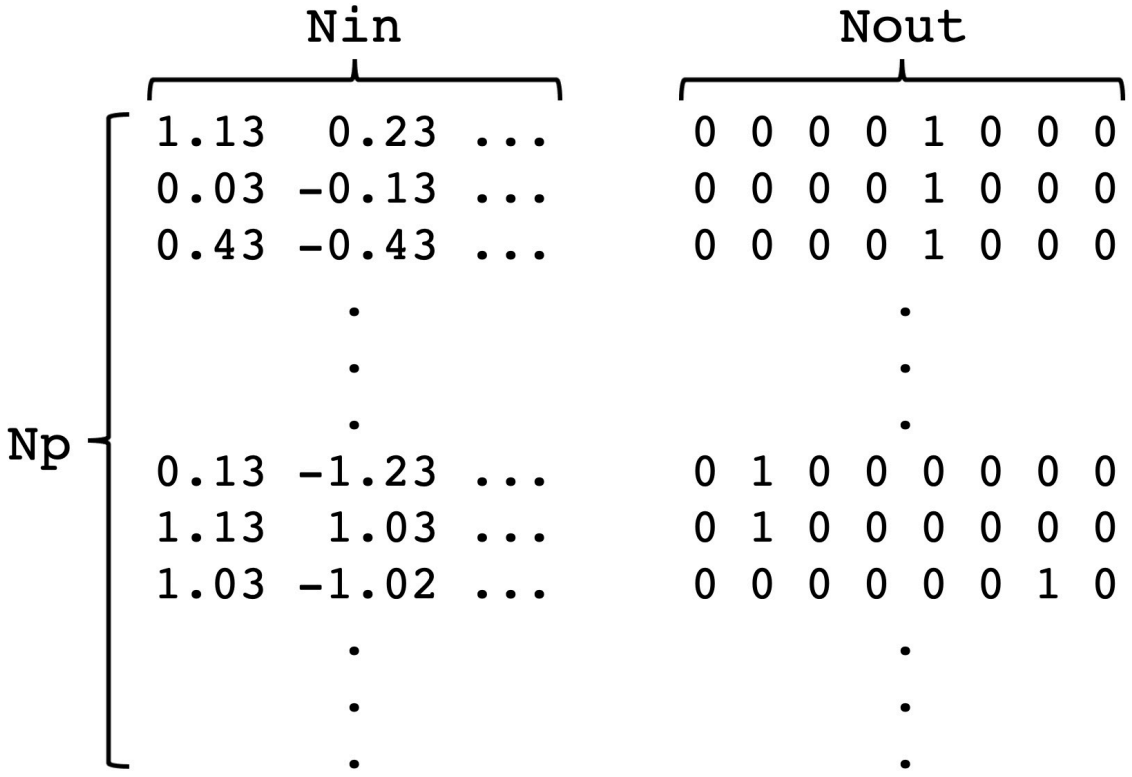
classification in MVPA

it looks like we have enough to train a classifier



classification in MVPA



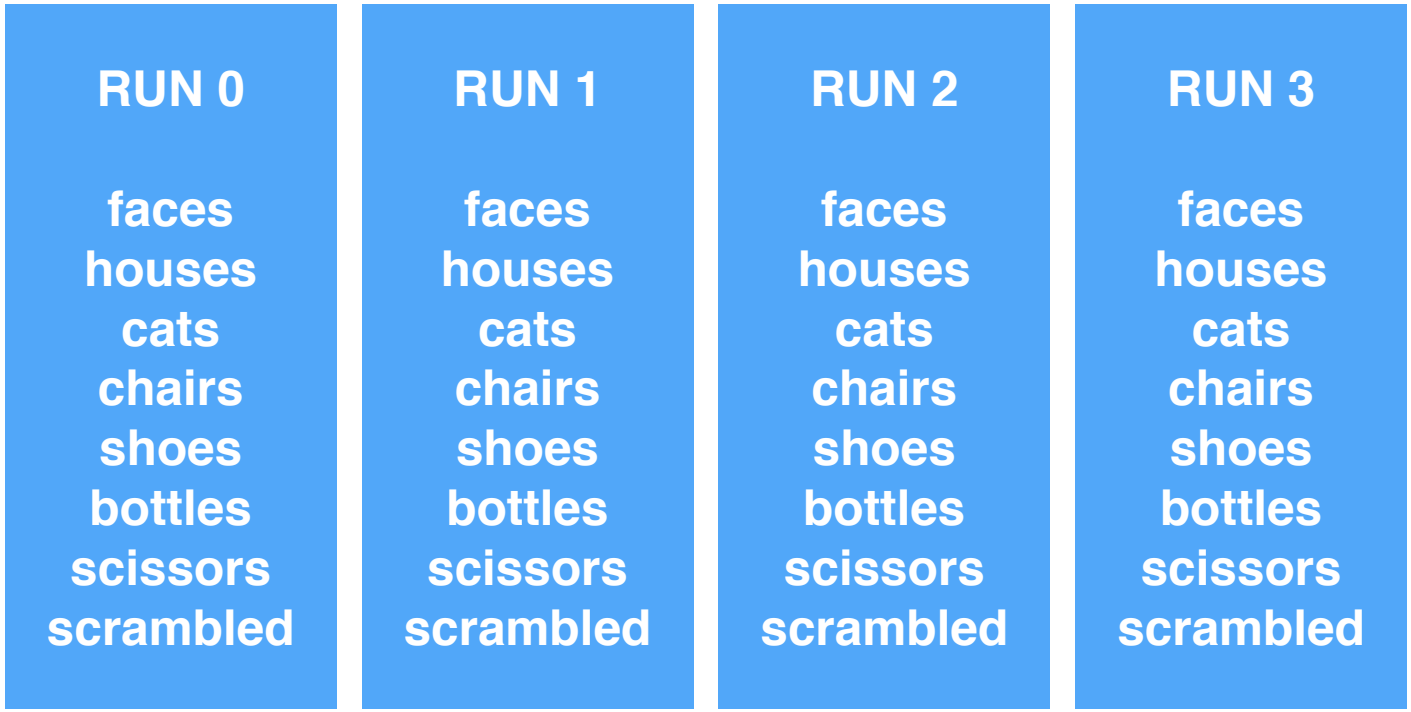


classification in MVPA

we're not training a classifier to classify some future data ...
we're using a classifier to measure neural activity, to ask whether there are patterns of voxel activity that predict object category

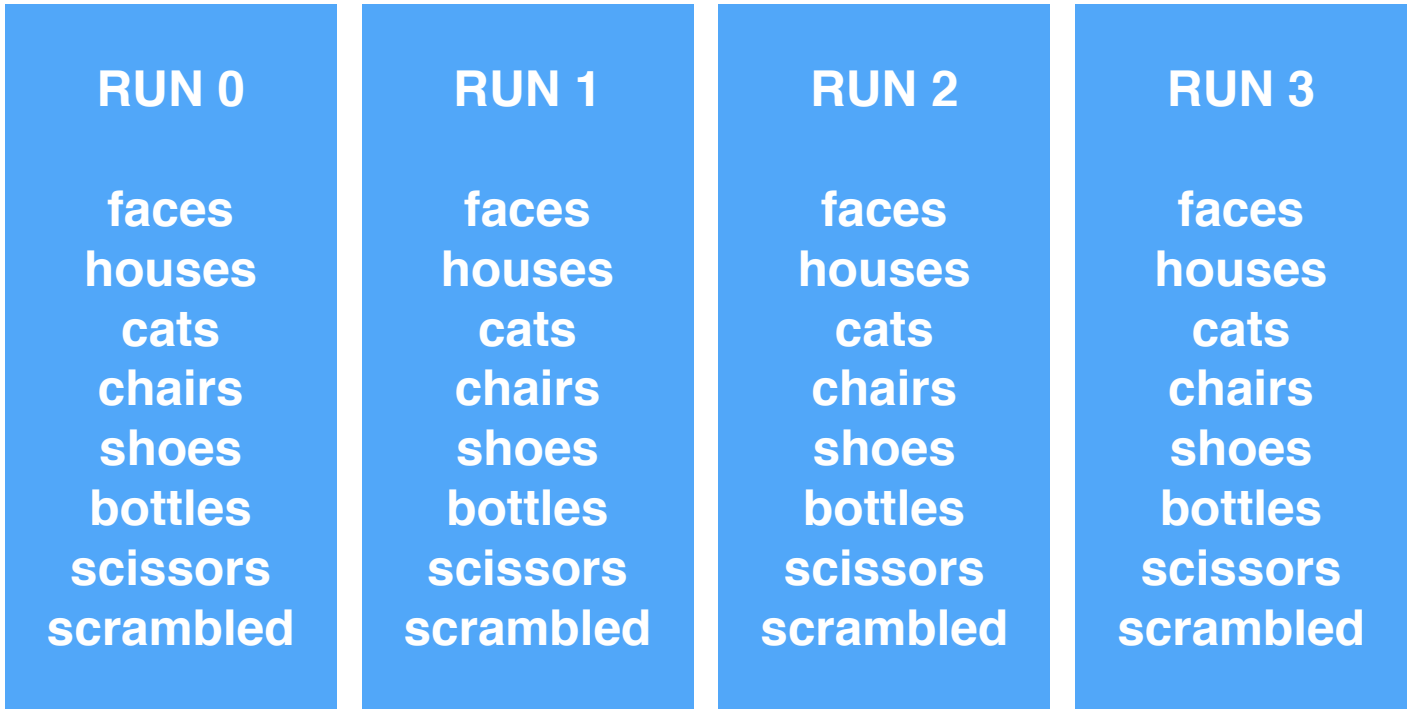
the classifier is a scientific (neural) measurement tool

classification in MVPA



let's keep things simple and assume 4 runs
(rather than the 12 runs we have)

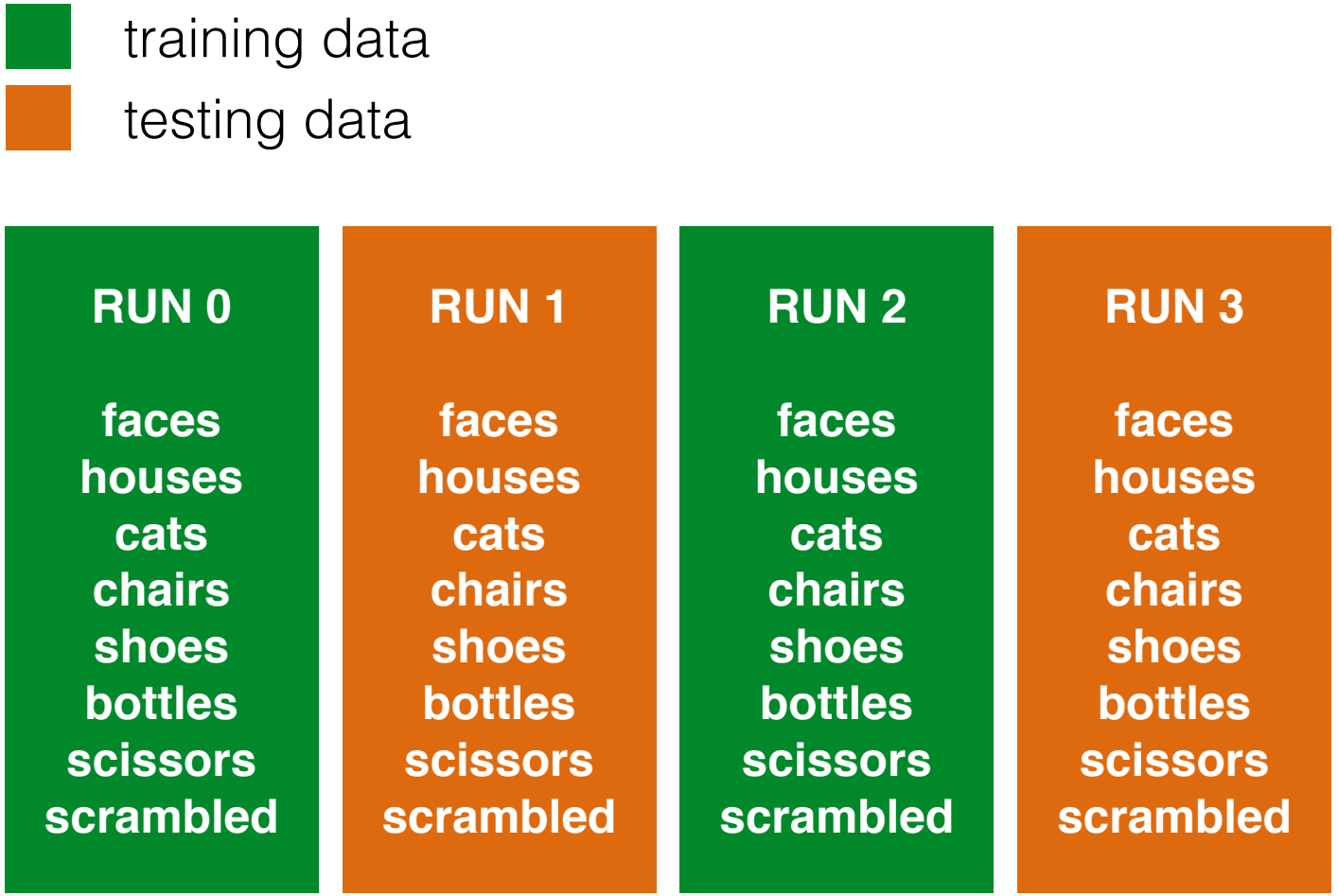
classification in MVPA



null hypothesis is that classifier performs at chance
(no pattern of neural activity predicts object category)

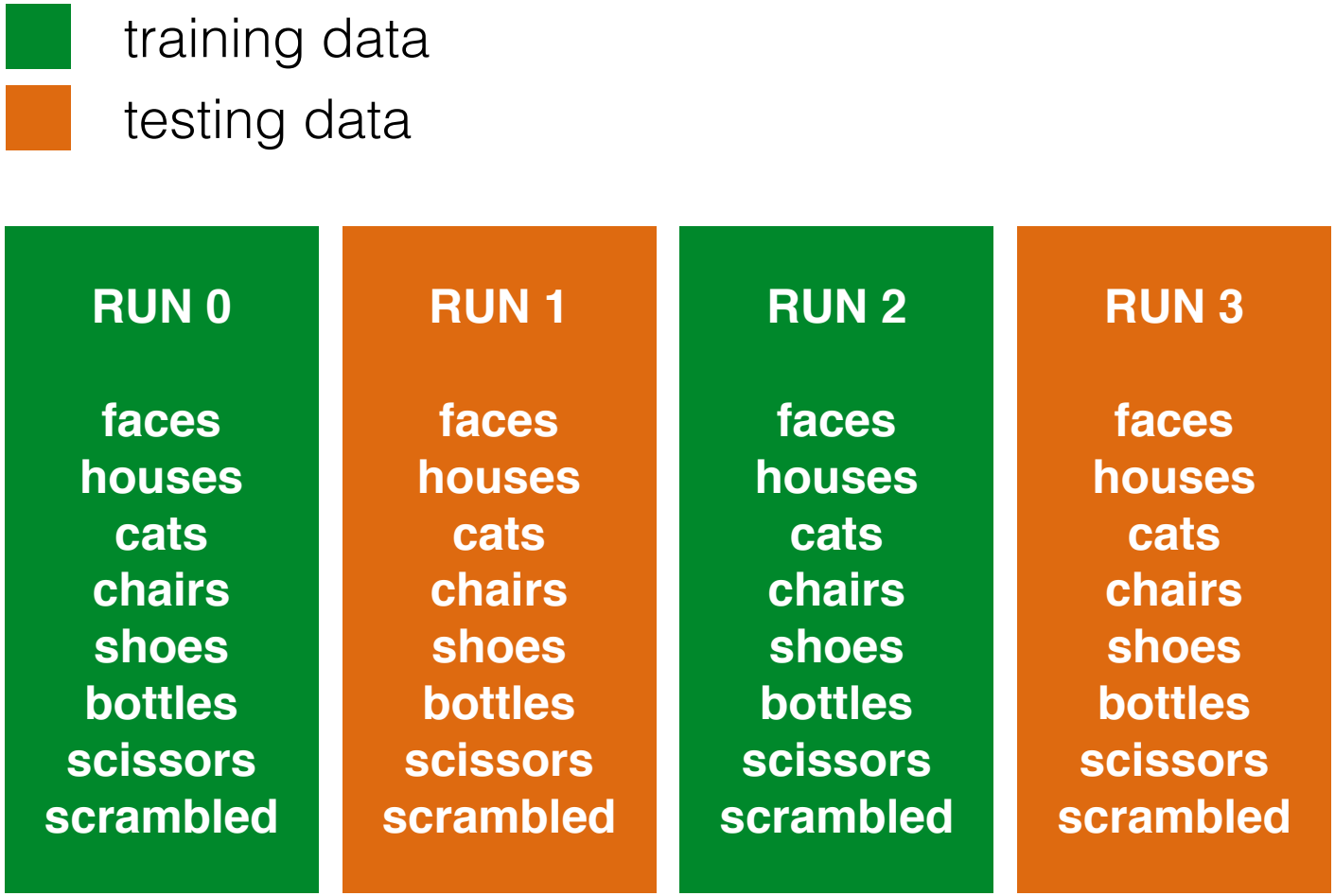
8 categories, chance is 1/8=12.5% classification

classification in MVPA



Haxby essentially used 1/2 the data (odd runs) to compare with other 1/2 of data (even runs) (with a "correlation classifier")

classification in MVPA



we could train a neural network classifier on 1/2 the data and test on the other 1/2 of the data

classification in MVPA

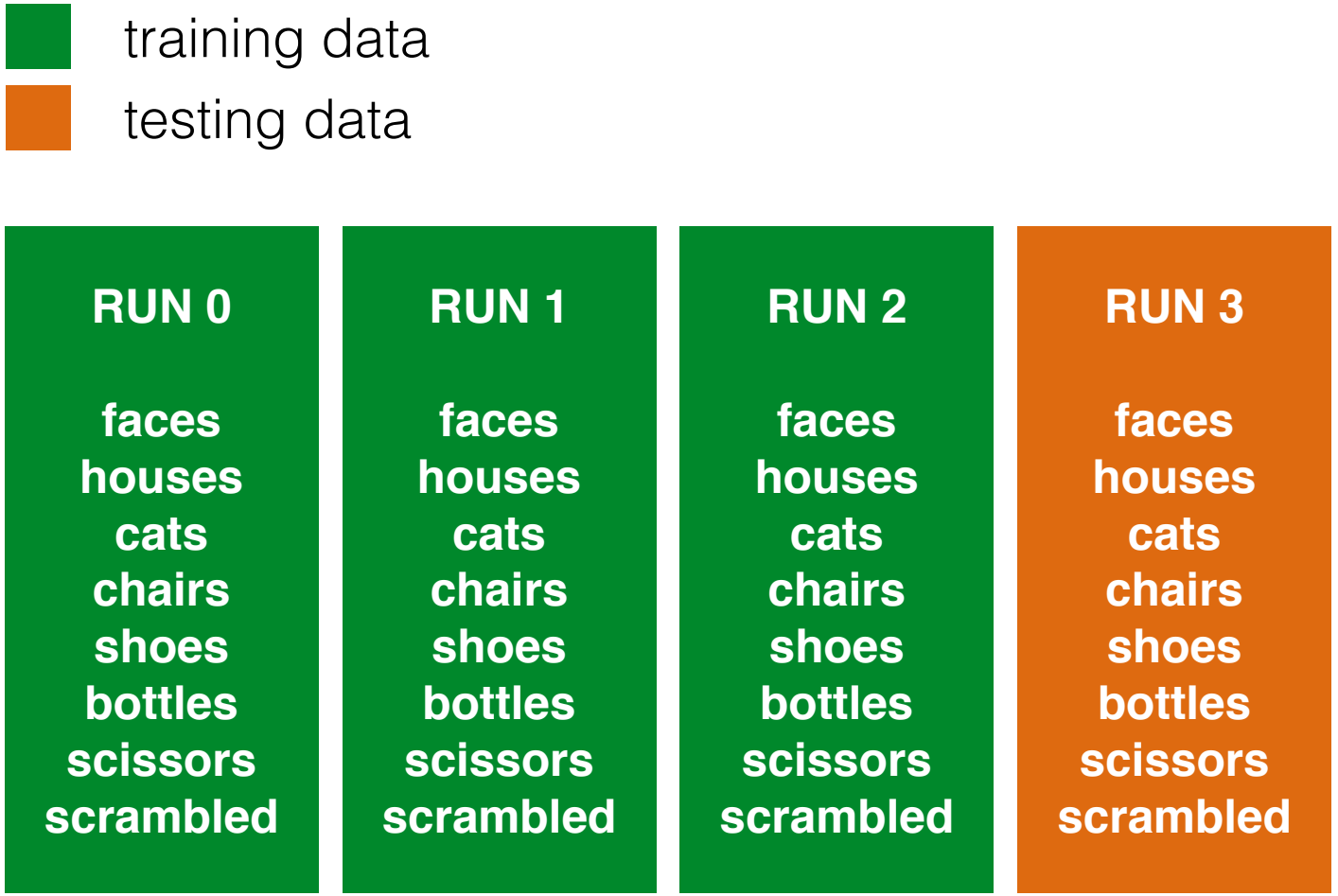
training data

testing data

RUN 0	RUN 1	RUN 2	RUN 3
faces	faces	faces	faces
houses	houses	houses	houses
cats	cats	cats	cats
chairs	chairs	chairs	chairs
shoes	shoes	shoes	shoes
bottles	bottles	bottles	bottles
scissors	scissors	scissors	scissors
scrambled	scrambled	scrambled	scrambled

or split up the data this way

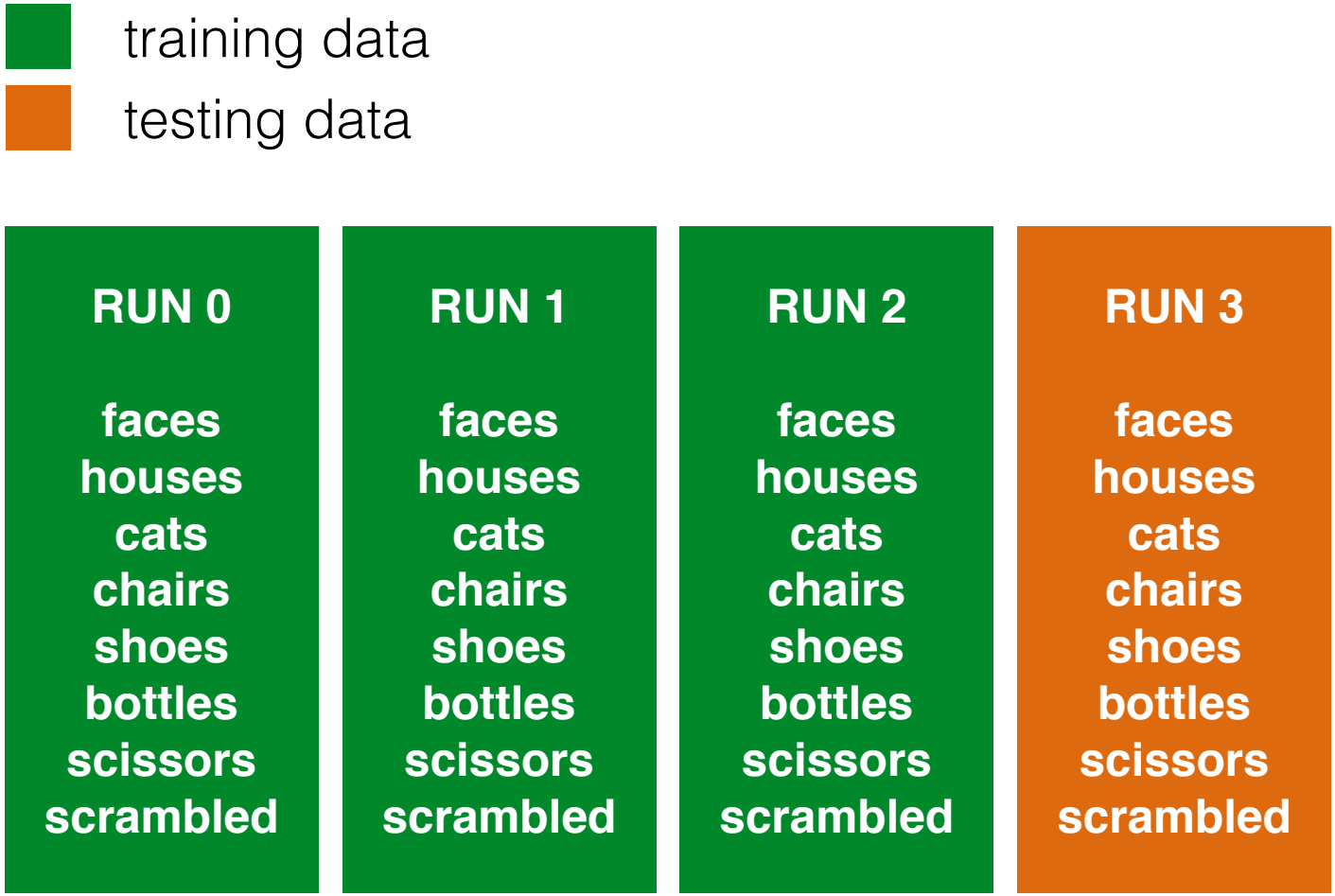
classification in MVPA



or train on 3/4 and test on a 1/4

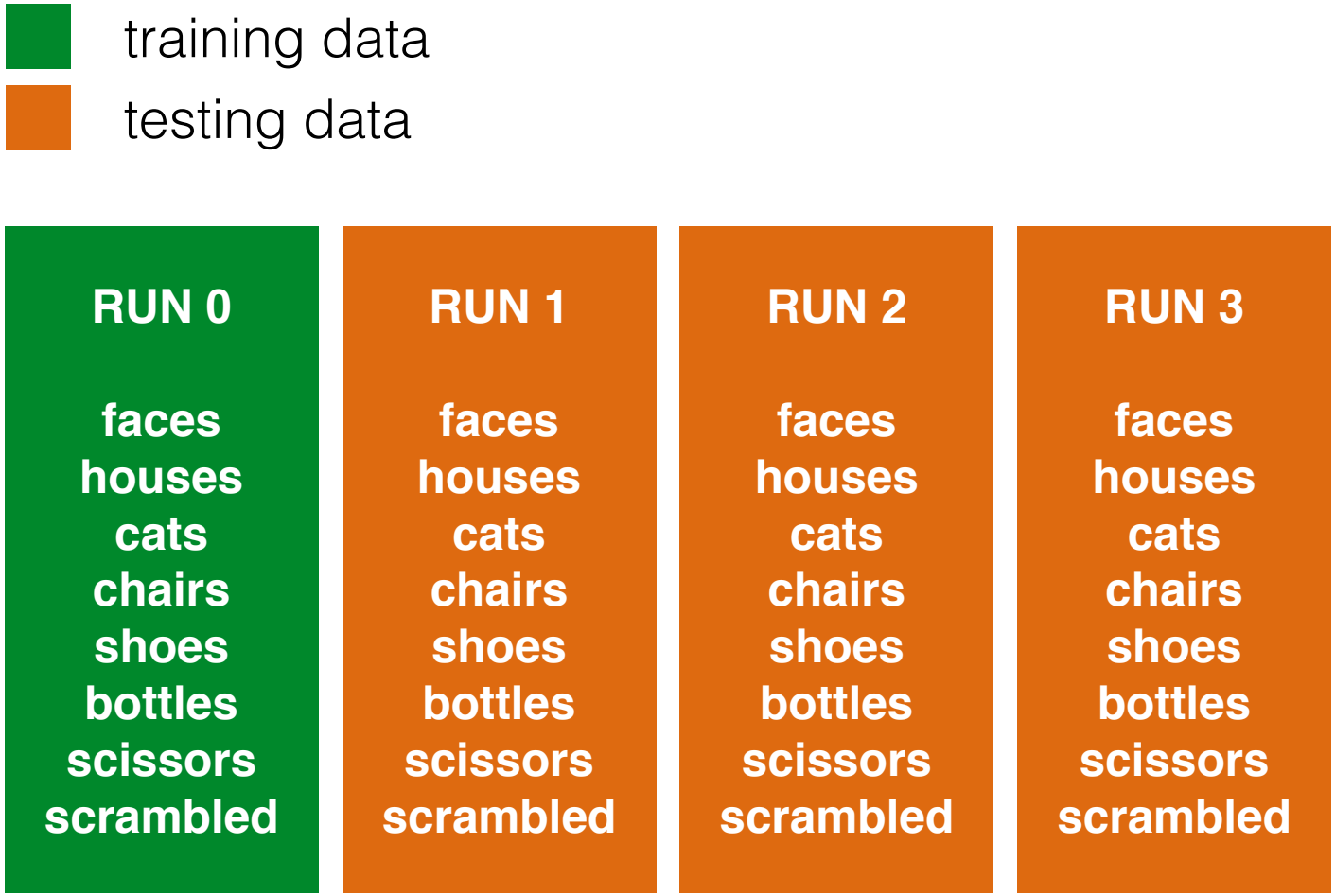
which is the best choice?

classification in MVPA



we don't have a lot of data, and the more training data the better, but that means less testing data

classification in MVPA



having more testing data is nice, but that means having less training data for the classifier

Homework8.ipynb

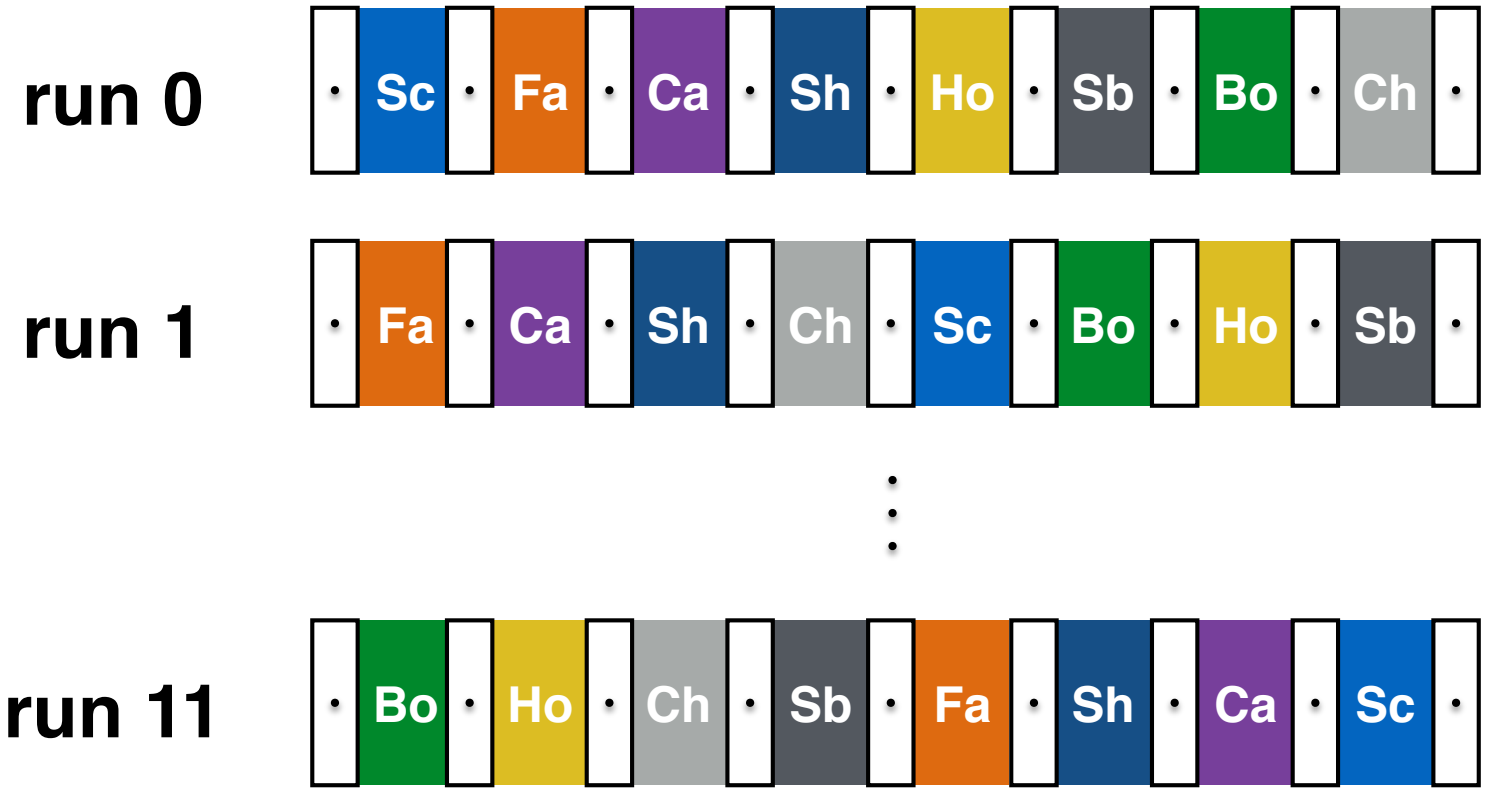
```
patterns, targets, runs = prep_haxby_data()
```

`prep_haxby_data()`

- opens `haxby_vt_patterns.npy`
- read in `labels.txt`
- "cheap version" of accounting for hemodynamic lag
- remove rests from runs

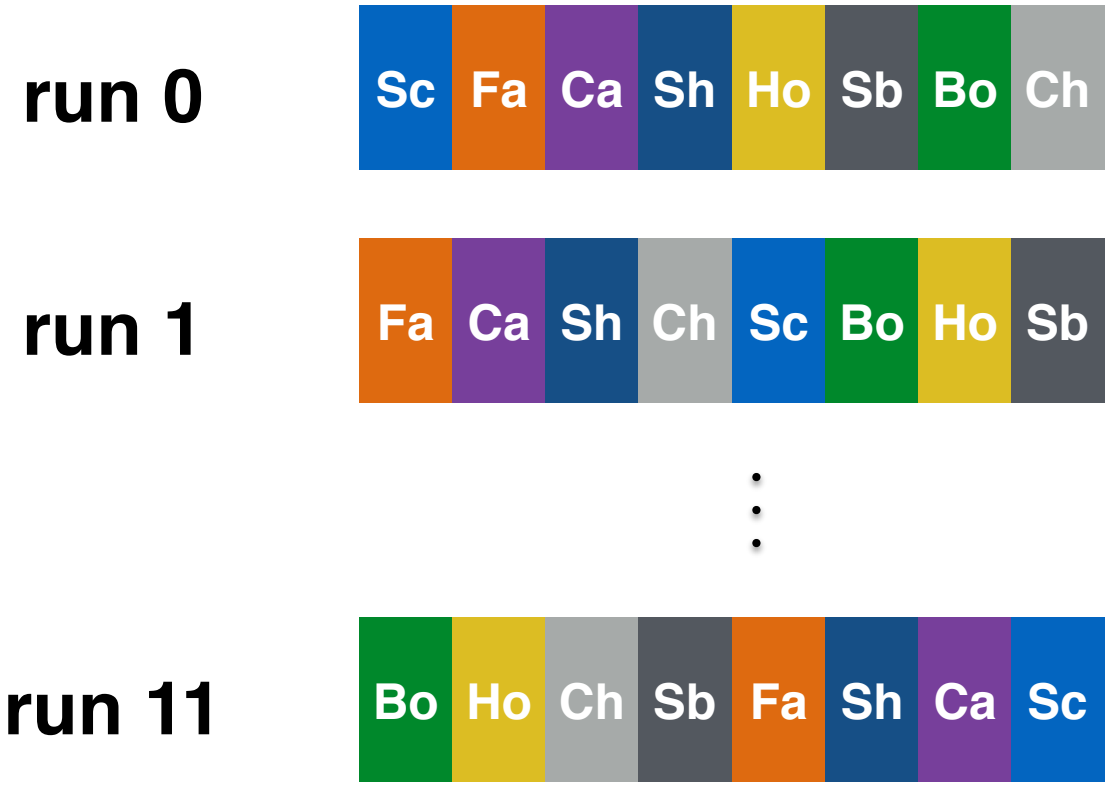
Haxby et al. 2001 details

raw data with rests



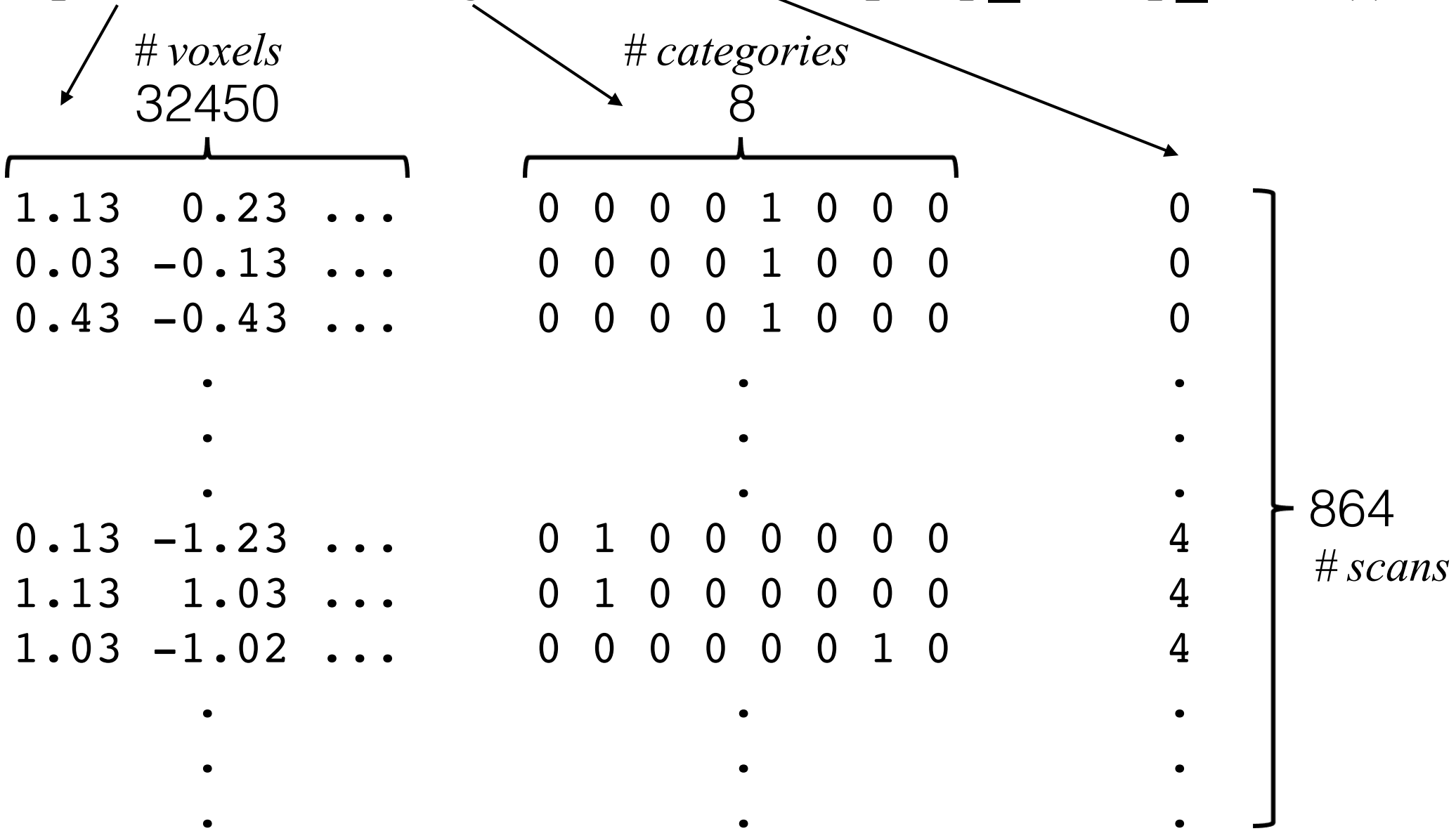
Haxby et al. 2001 details

with rests stripped away *from both labels and fMRI data*

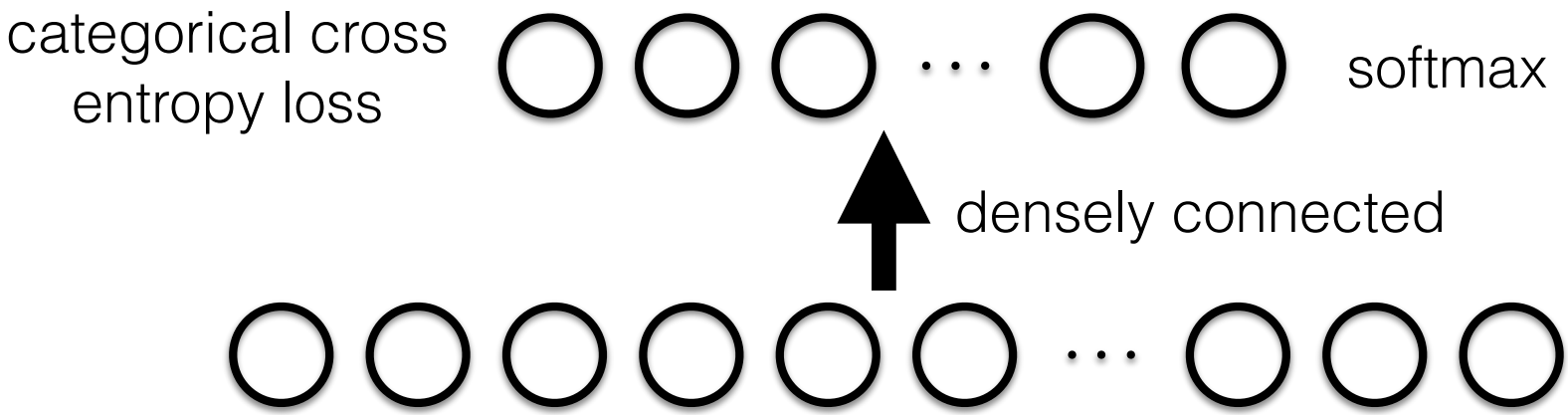


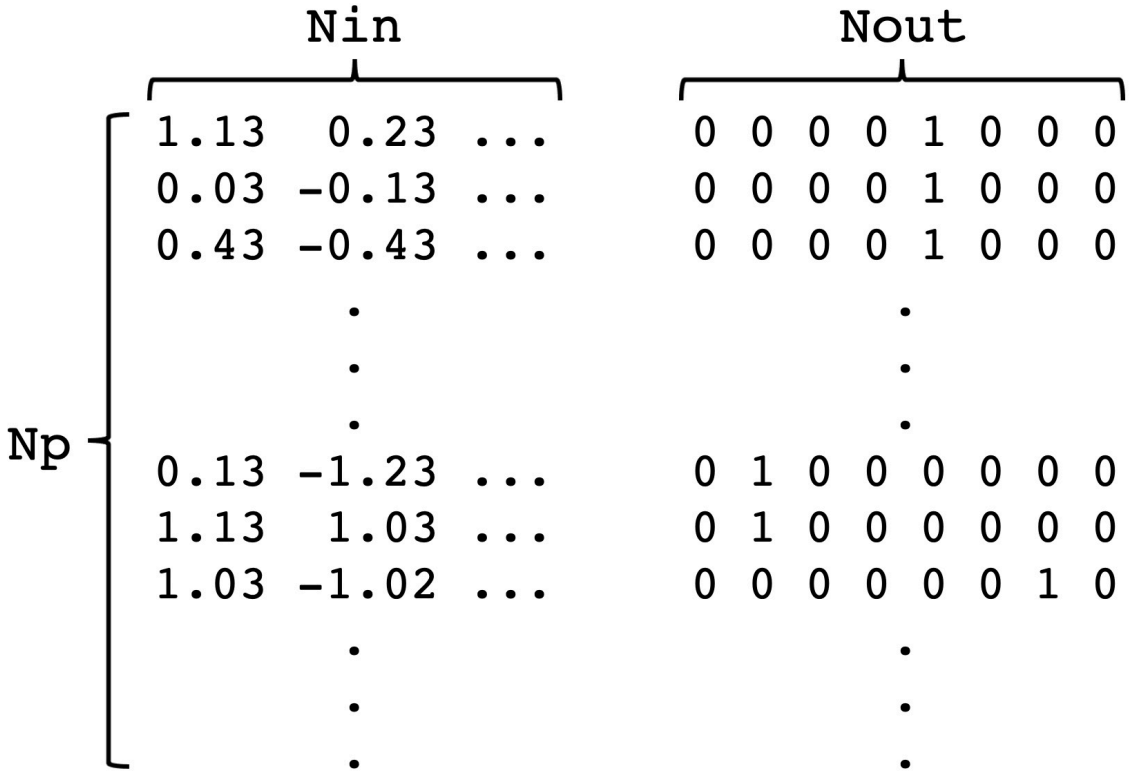
Homework8.ipynb

```
patterns, targets, runs = prep_haxby_data()
```



classification in MVPA

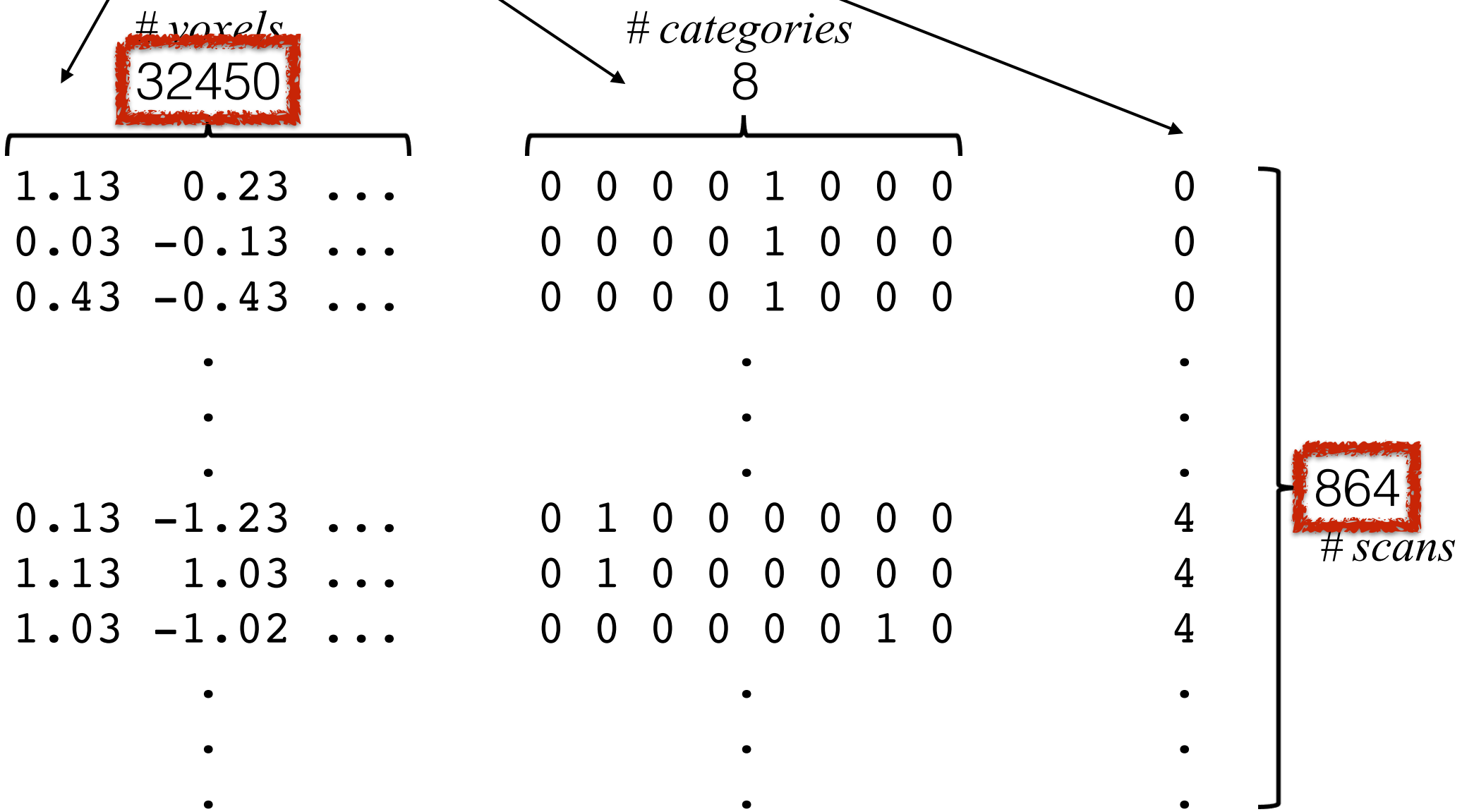




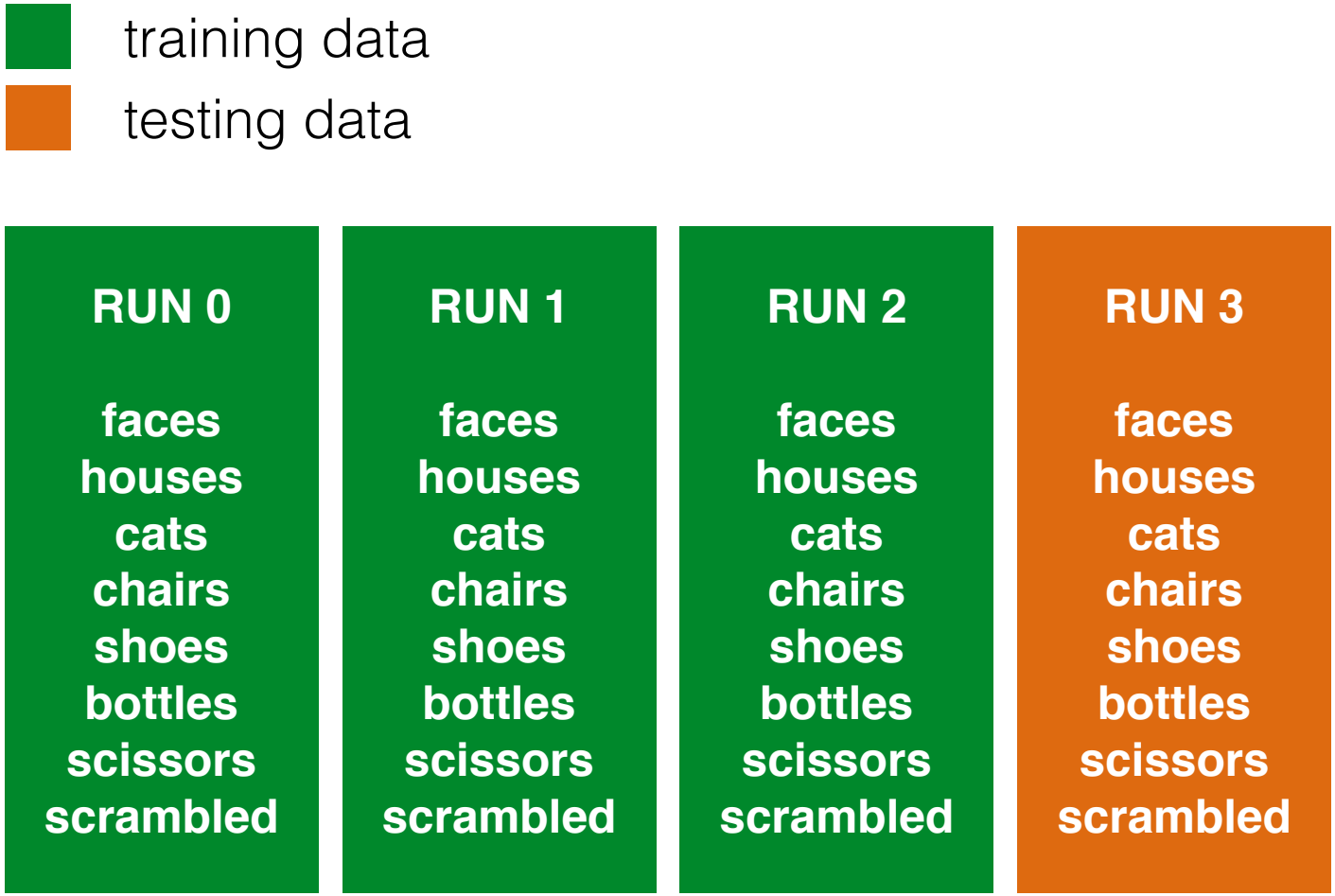
Homework8.ipynb

not much data to constrains lots of parameters

```
patterns, targets, runs = prep_haxby_data()
```

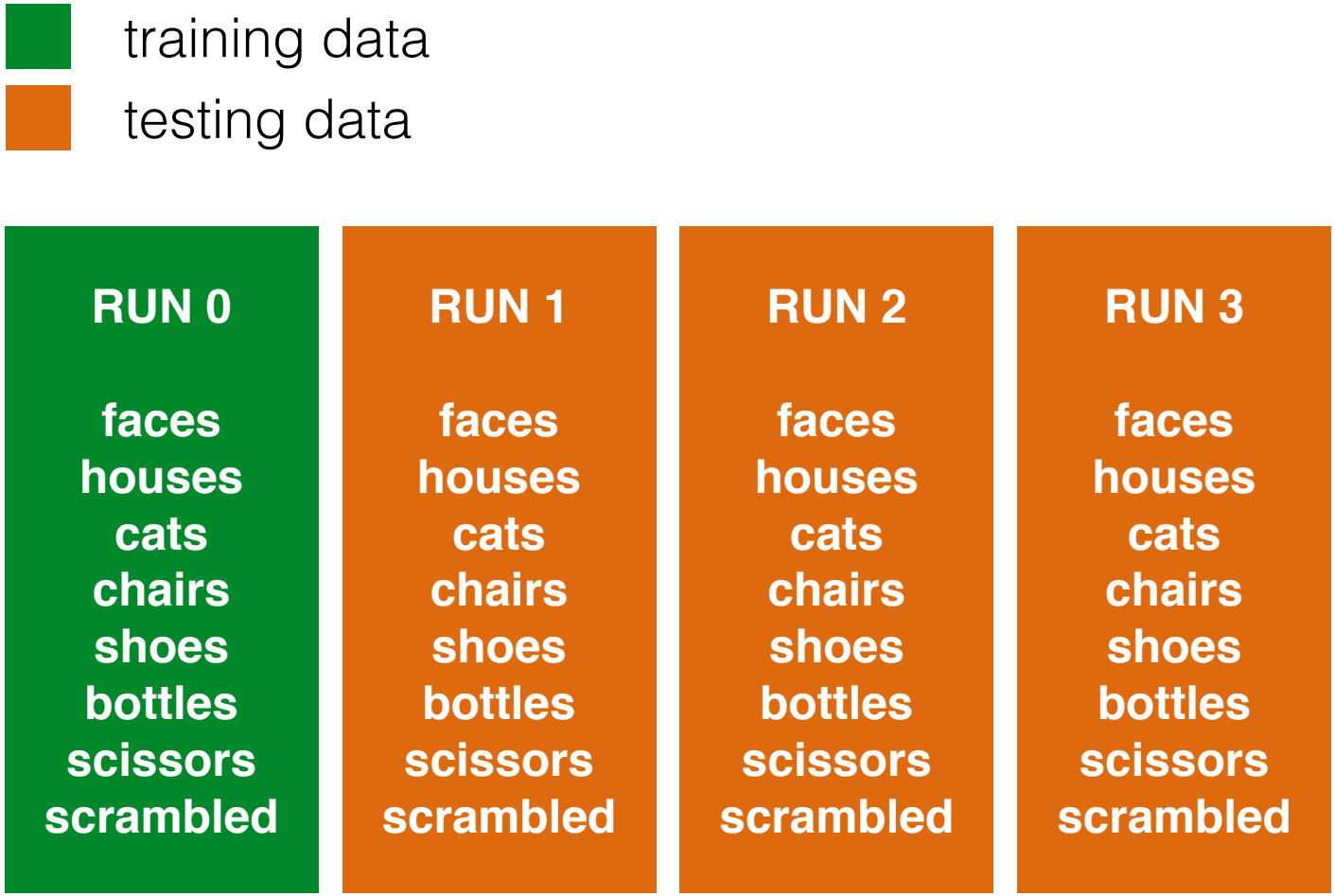


classification in MVPA



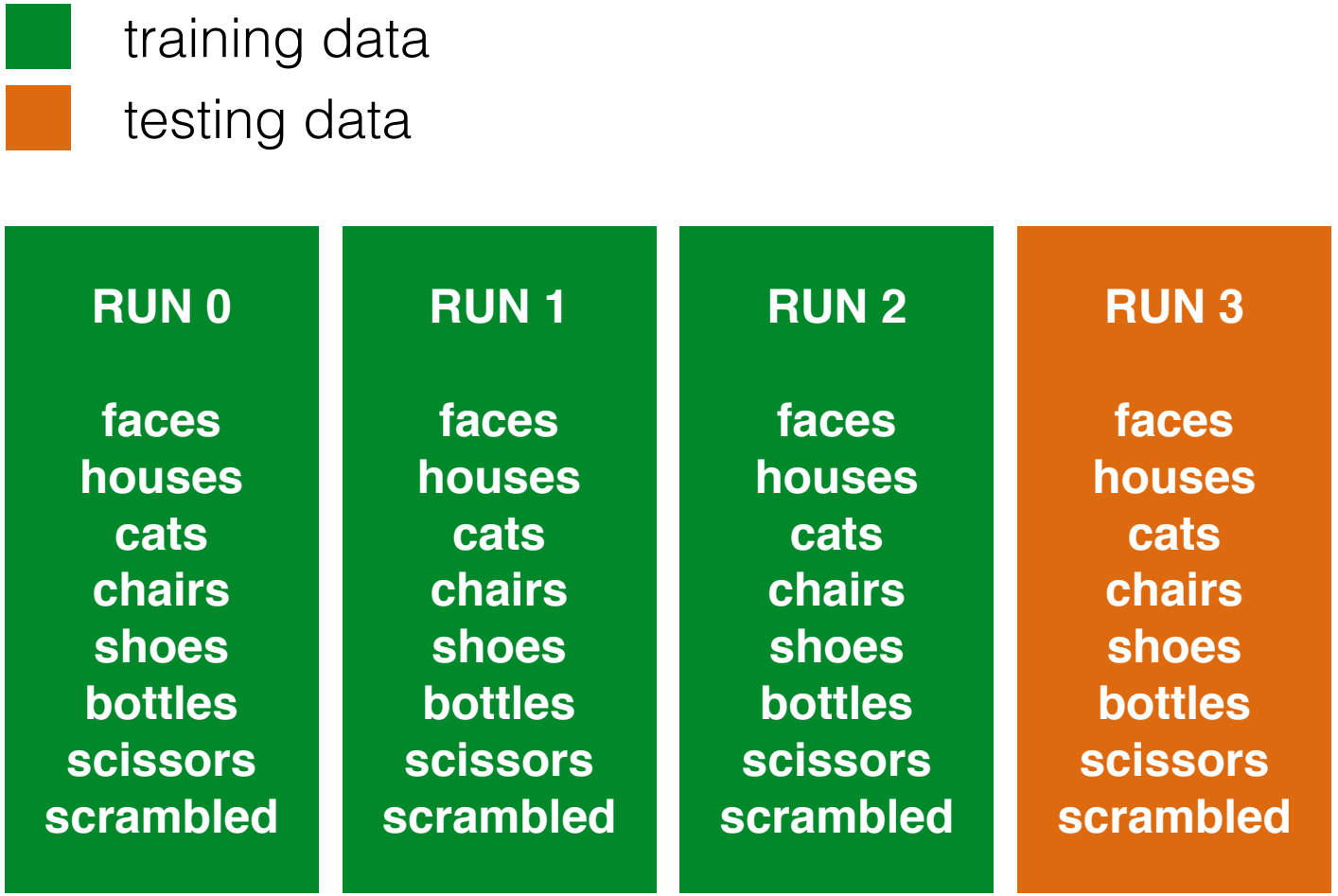
we don't have a lot of data, and the more training data the better, but that means less testing data

classification in MVPA



having more testing data is nice, but that means having less training data for the classifier

cross-validation



each iteration of cross-validation is called a "fold"

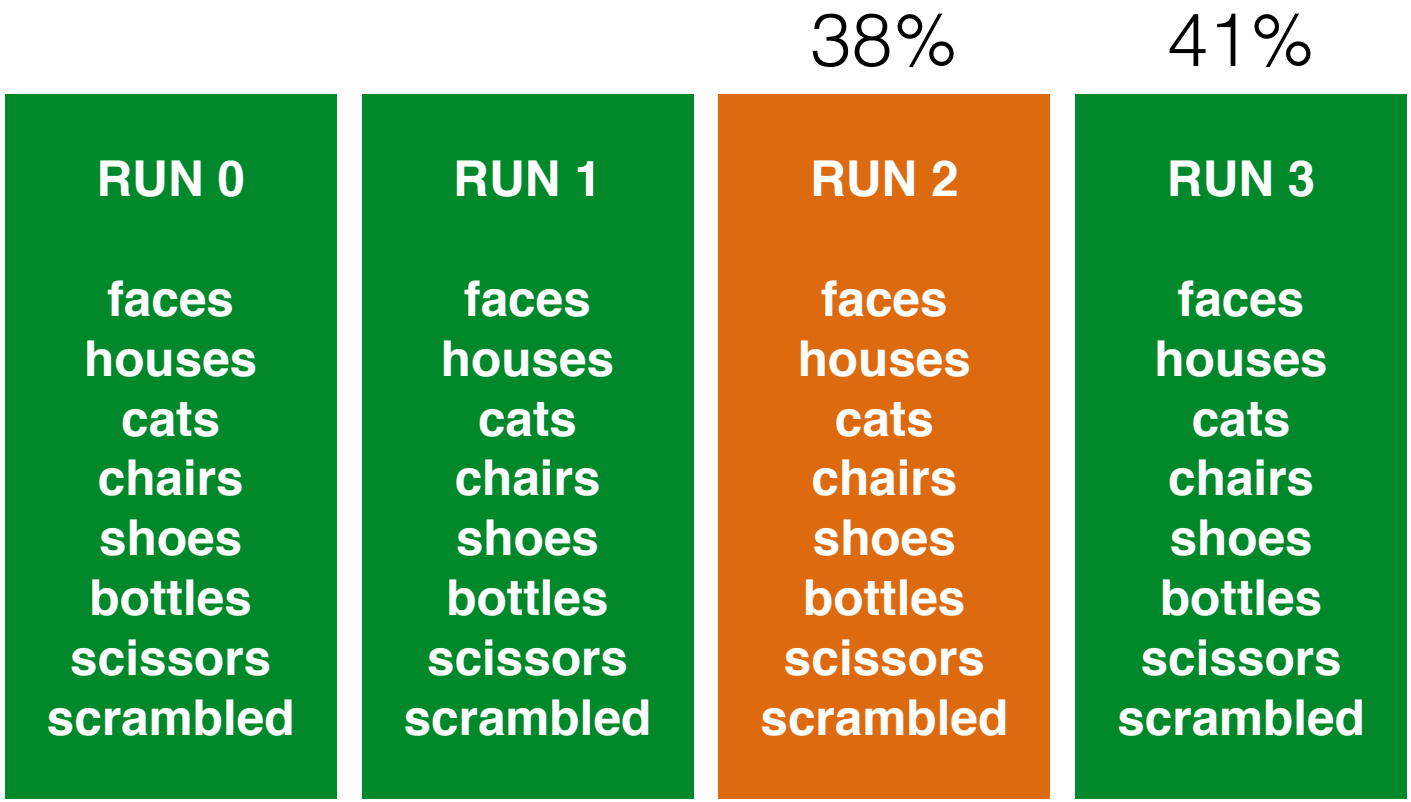
with 4 runs, we'd have 4 folds
(with 12 runs, we have 12 folds)

cross-validation

first fold: train on runs 0-3, test on run 4

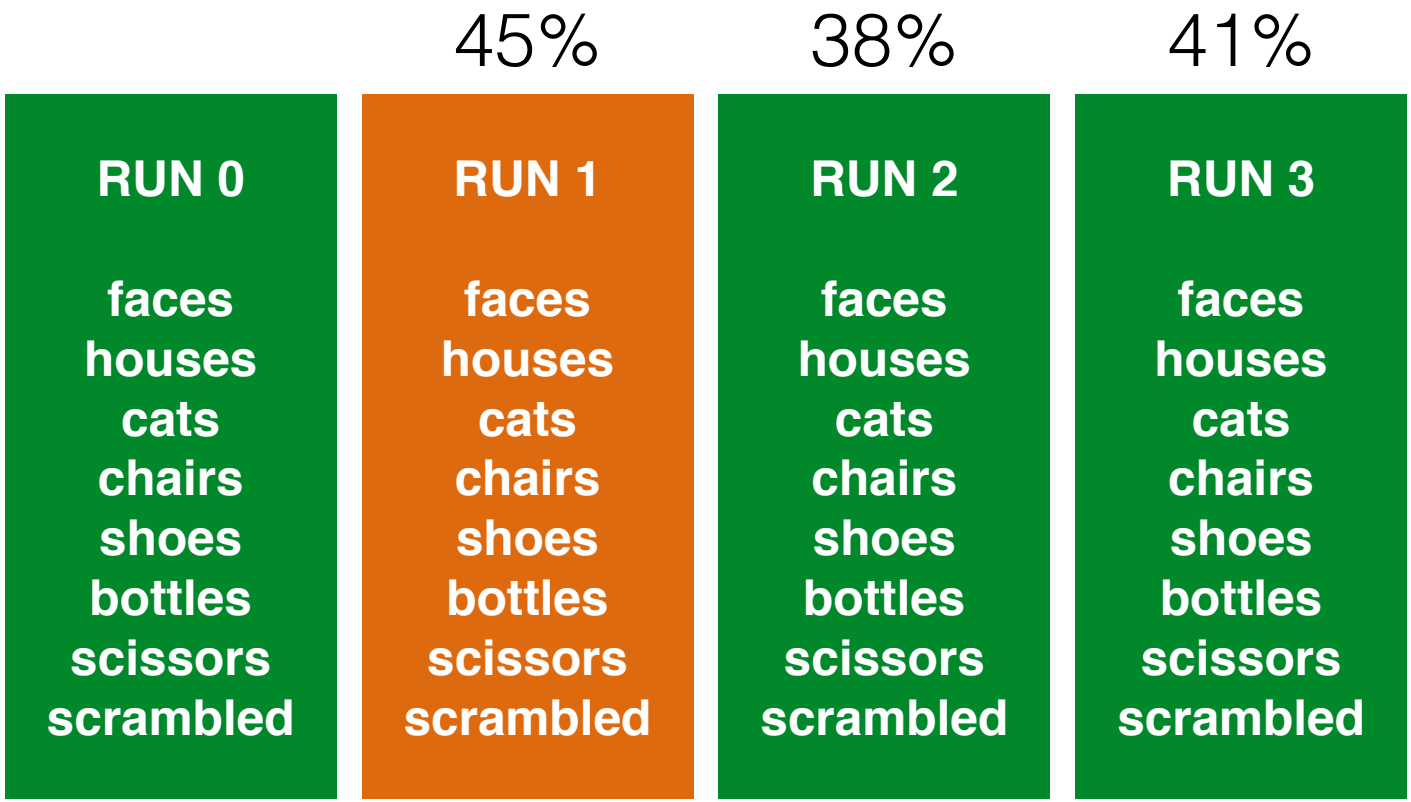
record classification accuracy for this fold

cross-validation



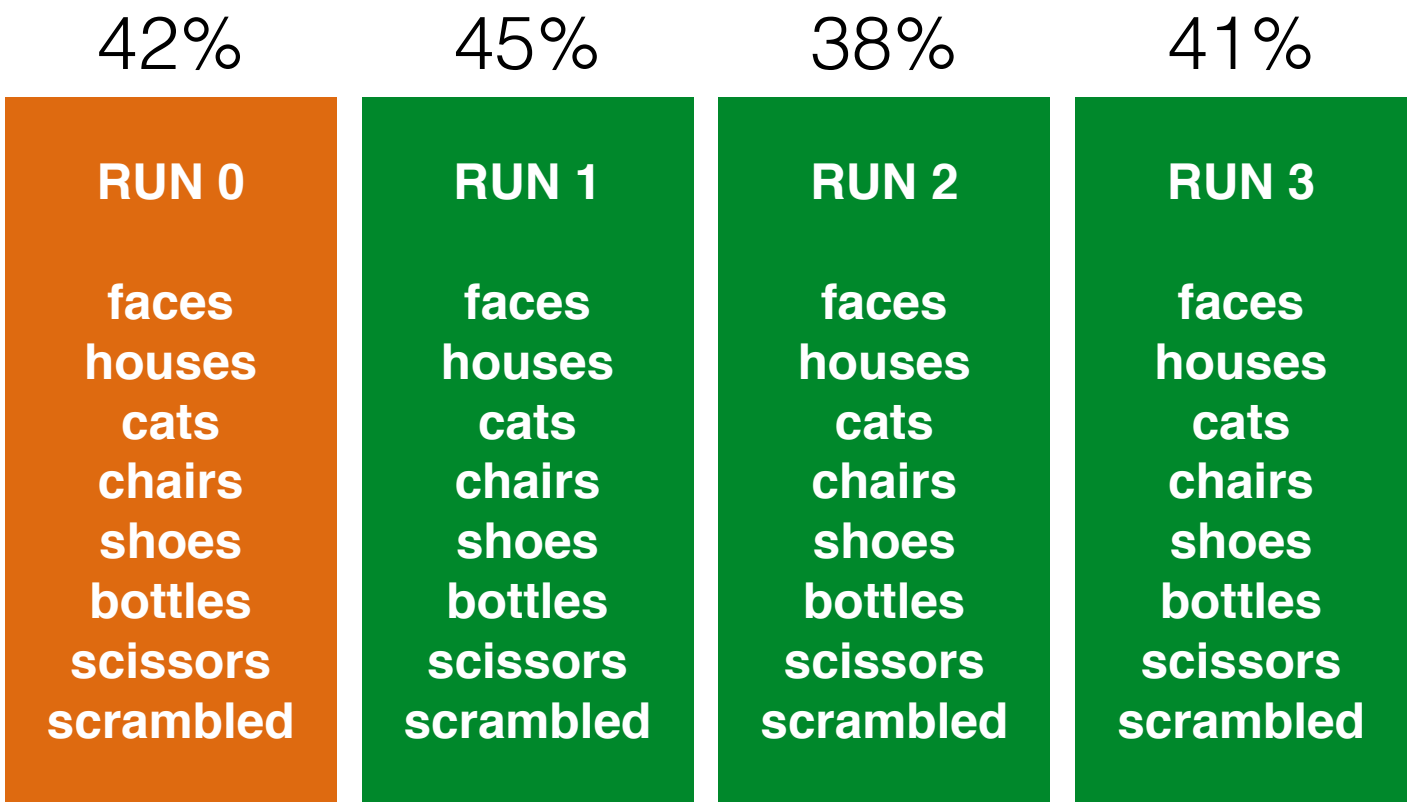
second fold

cross-validation



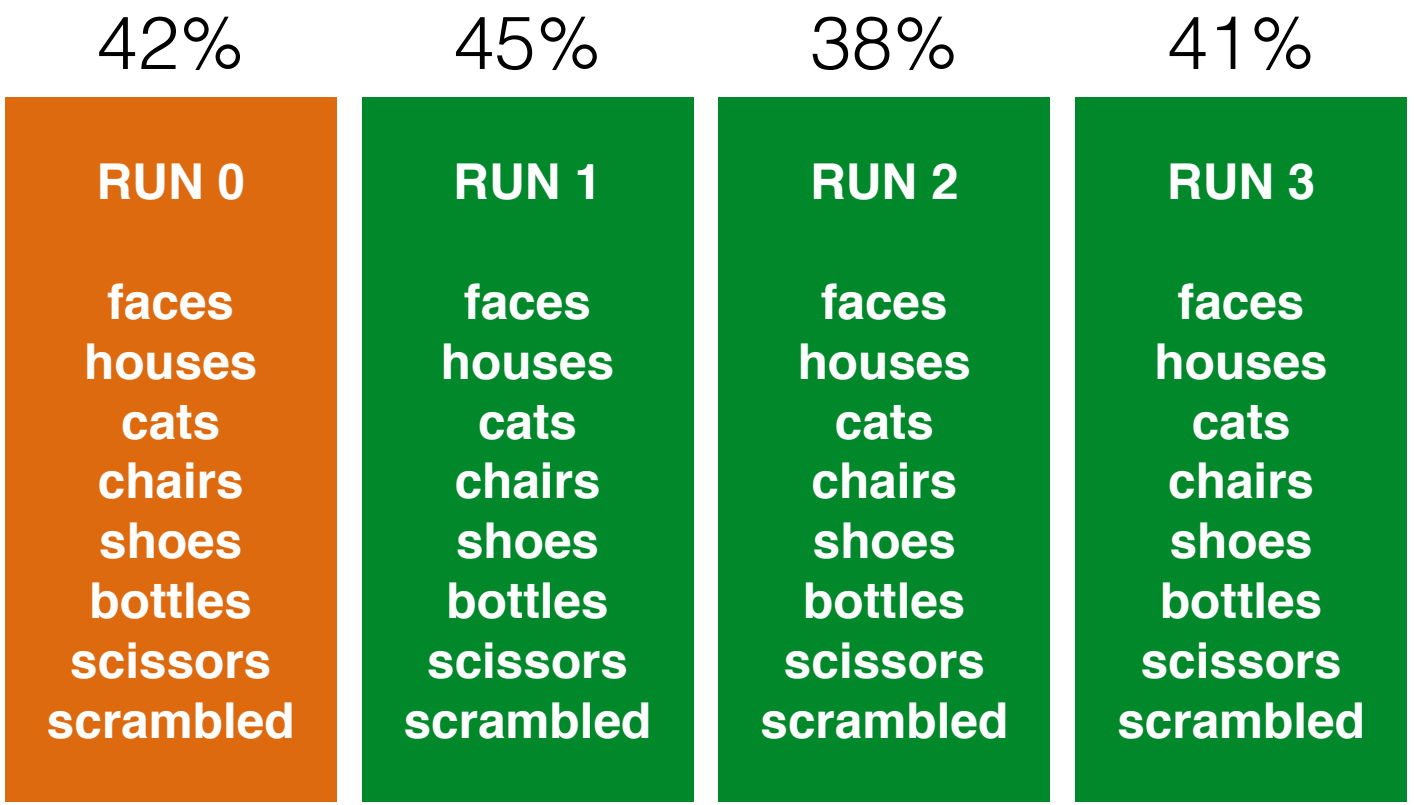
third fold

cross-validation



fourth fold

cross-validation



overall performance

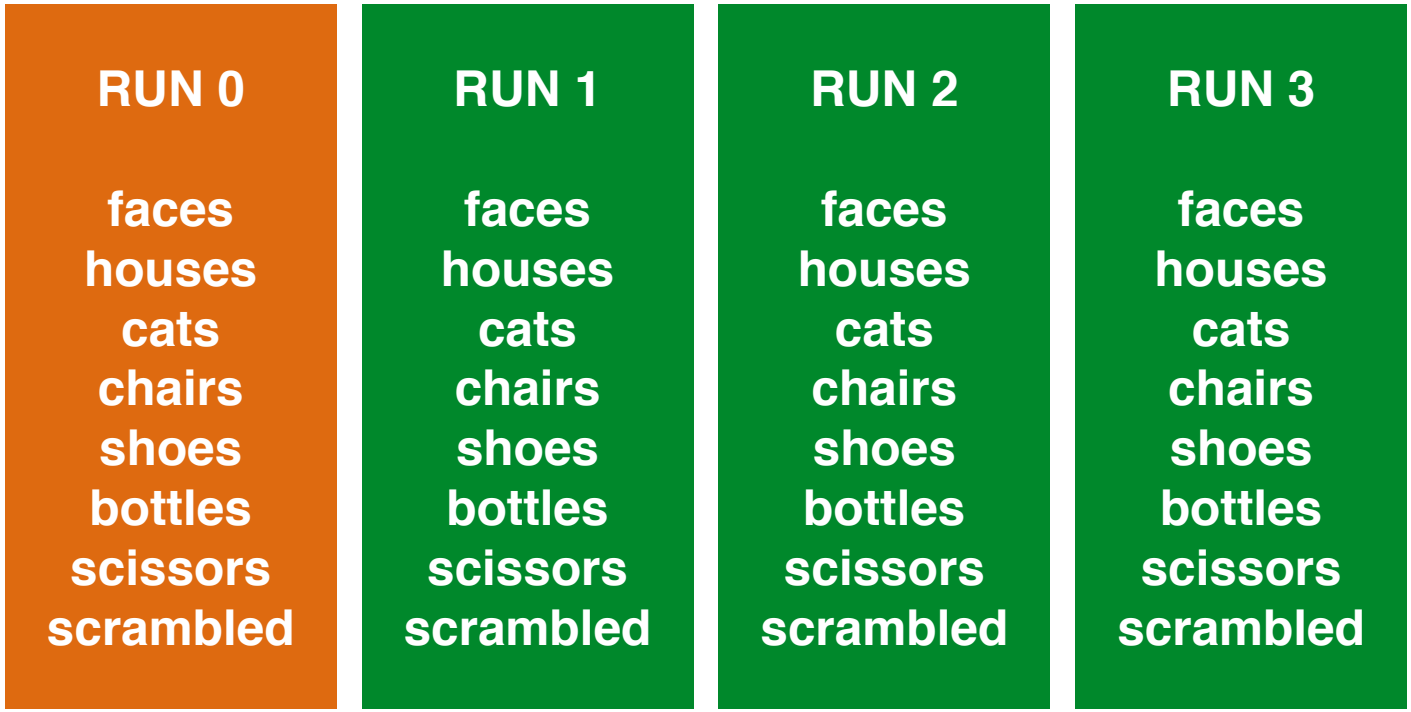
41.5%

(remember, chance is 12.5%)

cross-validation

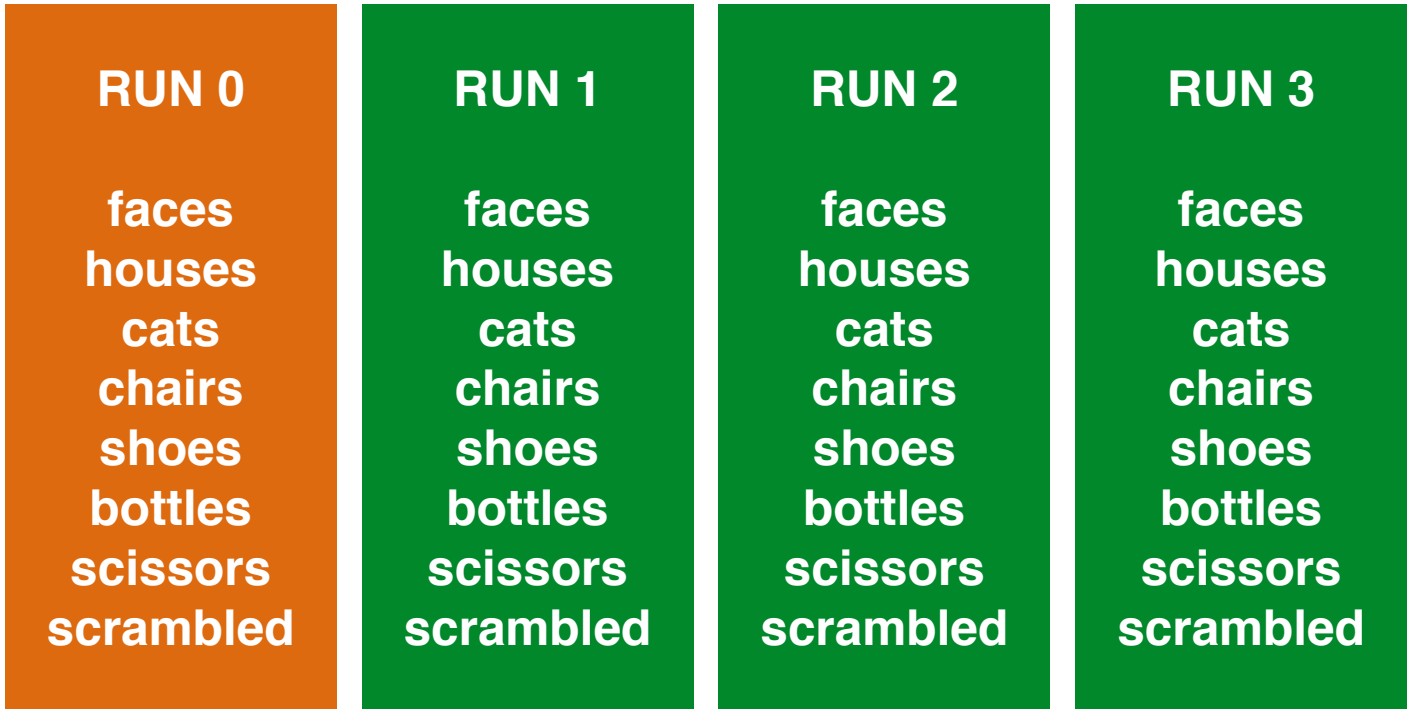
overall performance

41.5%



cross-validation essentially lets you use all of the data to both train and test classifiers

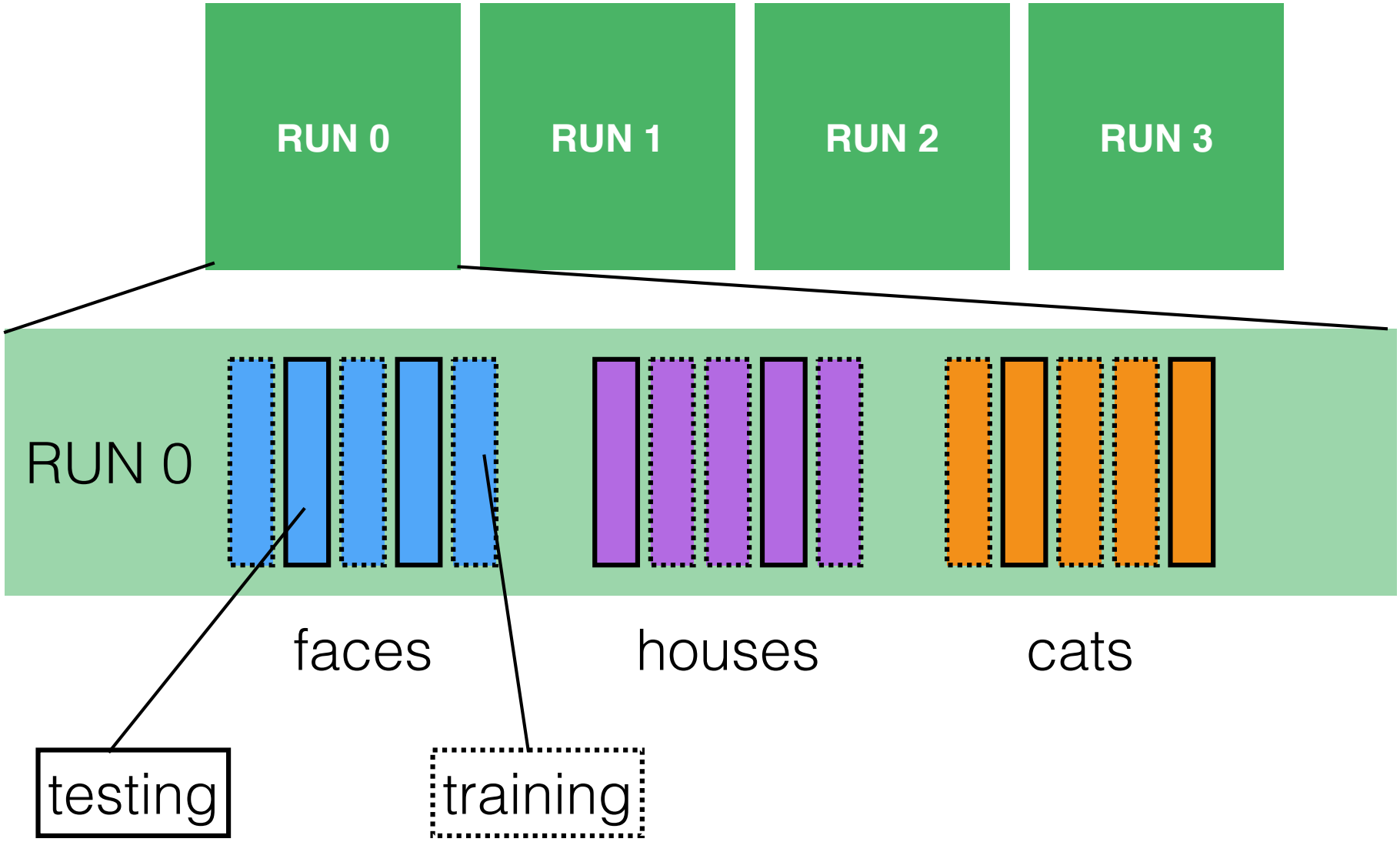
cross-validation with fMRI



note that here we're selecting folds based on complete runs (which is a recommended method for fMRI)

cross-validation with fMRI

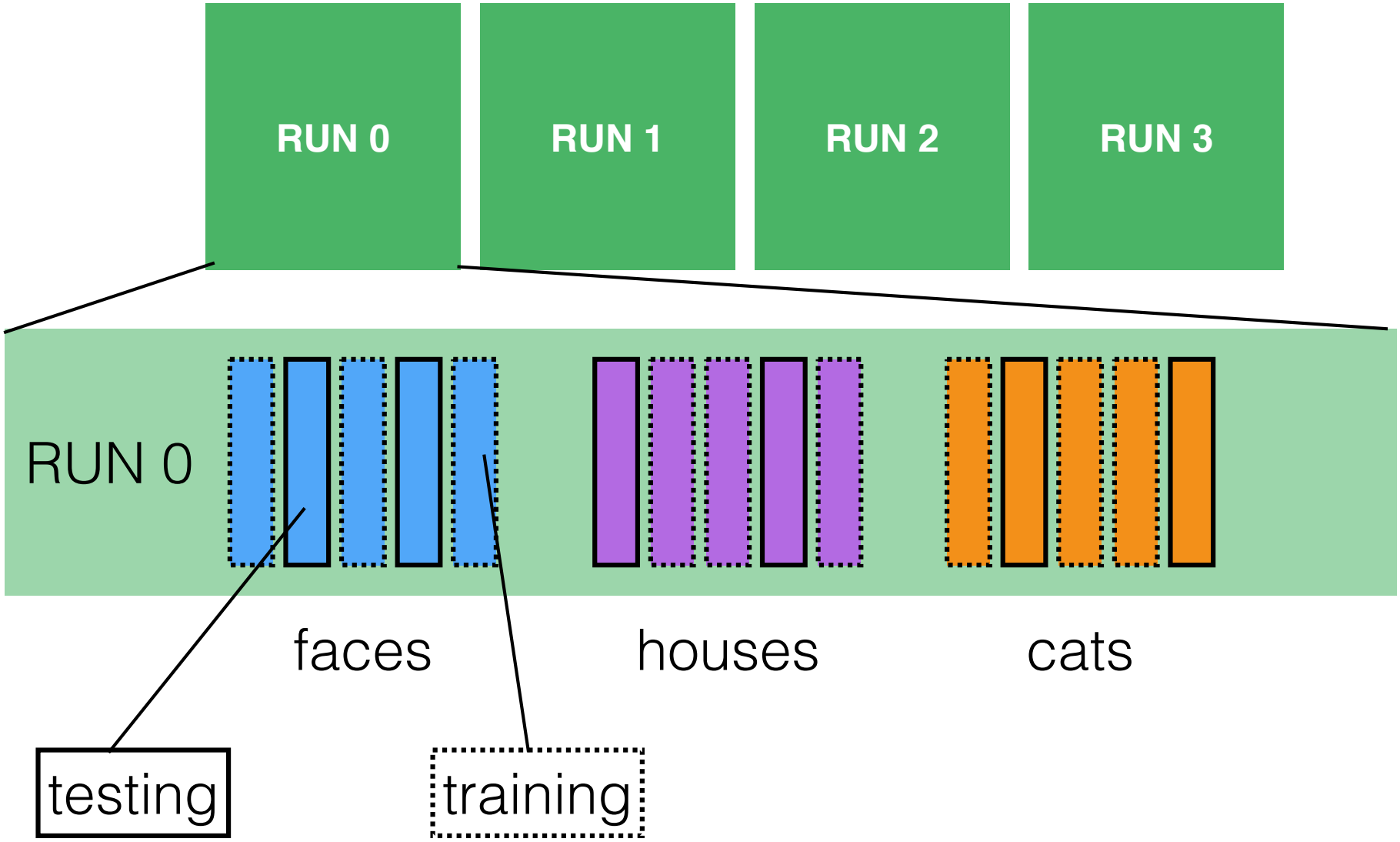
why not select folds within a run?



cross-validation with fMRI

(often from physical, non-physiological sources)

there is substantial "temporal autocorrelation" in fMRI data, which could spuriously inflate classifier performance



Homework 8 (PART ONE)

implement a 12-fold cross-validation of **MVPA** using a **softmax classifier**

do patterns of voxel activity predict object category

single-layer neural network with softmax and categorical cross entropy

set up data structures to store classifier performance

set up data structures to store classifier performance

```
for r in range(n_runs):
```

loop over the 12 folds

set up data structures to store classifier performance

```
for r in range(n_runs):
```

set up the fold r

set up data structures to store classifier performance

```
for r in range(n_runs):
```

set up the fold r

do feature selection for this fold

set up data structures to store classifier performance

```
for r in range(n_runs):
```

set up the fold r

do feature selection for this fold

train classifier for this fold

set up data structures to store classifier performance

```
for r in range(n_runs):
```

set up the fold r

do feature selection for this fold

train classifier for this fold

test classifier for this fold

set up data structures to store classifier performance

```
for r in range(n_runs):
```

- set up the fold r
- do feature selection for this fold
- train classifier for this fold
- test classifier for this fold
- save classifier performance

set up data structures to store classifier performance

```
for r in range(n_runs):
```

- set up the fold r
- do feature selection for this fold
- train classifier for this fold
- test classifier for this fold
- save classifier performance

report average of classifiers

set up data structures to store classifier performance

```
for i in range(n_iters):
    for r in range(n_runs):
```

do this many times (since neural networks stochastic)

set up the fold r

do feature selection for this fold

train classifier for this fold

test classifier for this fold

save classifier performance

report average of classifiers

set up data structures to store classifier performance

```
for i in range(n_iters):
    for r in range(n_runs):
```

in practice, `n_iters` = 100
for Homework, `n_iters` = 5

set up the fold r

do feature selection for this fold

train classifier for this fold

test classifier for this fold

save classifier performance

report average of classifiers

set up data structures to store classifier performance

```
for i in range(n_iters):
    for r in range(n_runs):
```

set up the fold r

do feature selection for this fold

train classifier for this fold

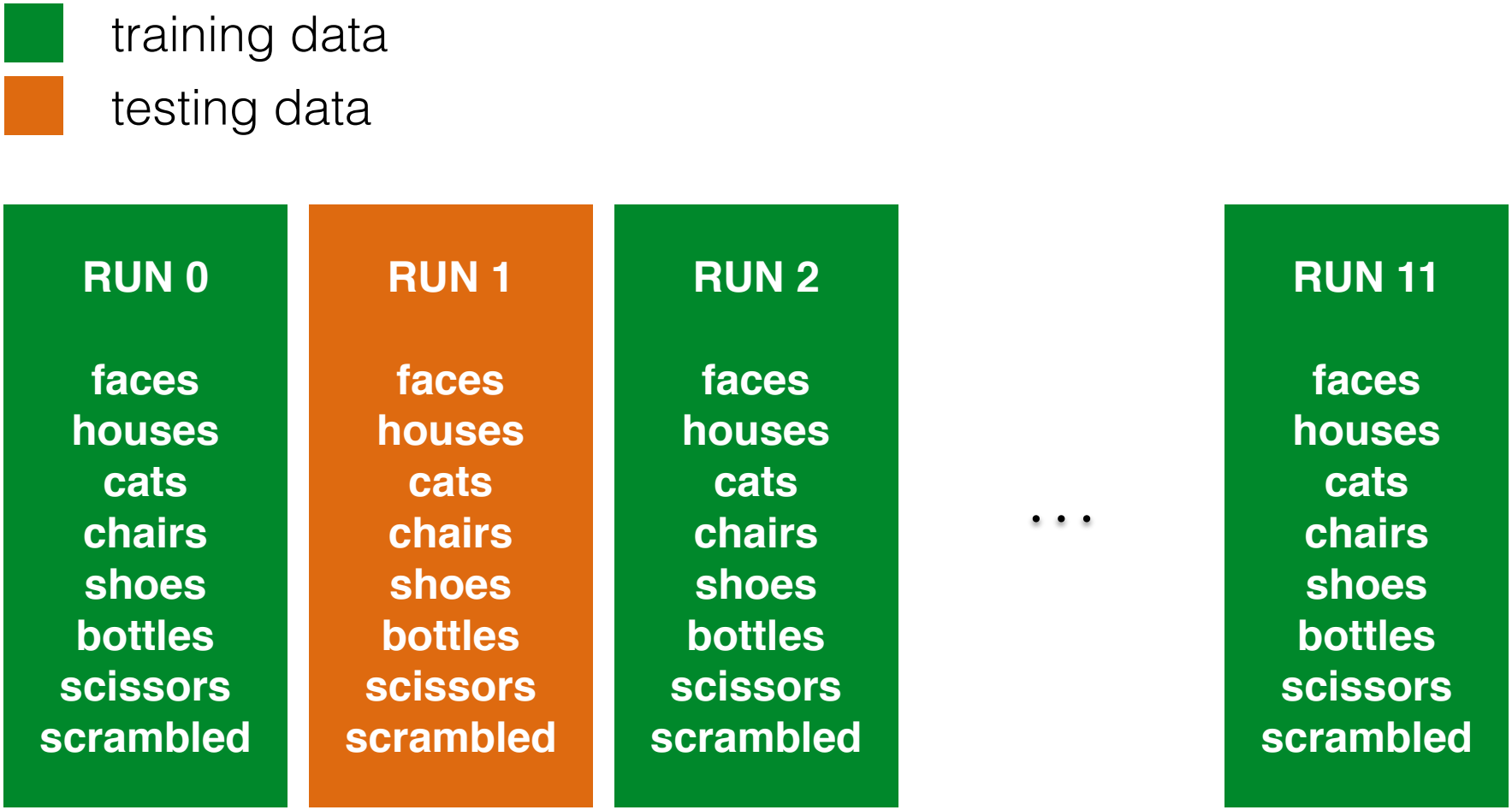
test classifier for this fold

save classifier performance

report average of classifiers

make each of these a separate function

set up the fold r



for example, suppose we're leaving out run 1

set up the fold r

see **Homework8.ipynb**

set up the fold r

```
# an example "leaving out" a particular run

example_test_run = 1

train_these = runs != example_test_run
test_these  = runs == example_test_run

# an example of logical indexing
train_patterns = patterns[train_these, :]
train_targets = targets[train_these, :]

test_patterns = patterns[test_these, :]
test_targets = targets[test_these, :]
```

do feature selection for this fold

```
fs_train_patterns, fs_test_patterns =
                       feature_selection(train_patterns,
                                         train_targets,
                                         test_patterns)
```

these will have different numbers of voxels depending on run

train classifier for this fold

- create classifier in Keras
  - softmax activation function
  - categorical cross entropy for loss function

train classifier for this fold

- create classifier in Keras
  - softmax activation function
  - categorical cross entropy for loss function
  - use no validation patterns
    - *too little data to set aside validation patterns*
    - *not creating a classification "engine"*
    - *cross-validation fills role of evaluating*

train classifier for this fold

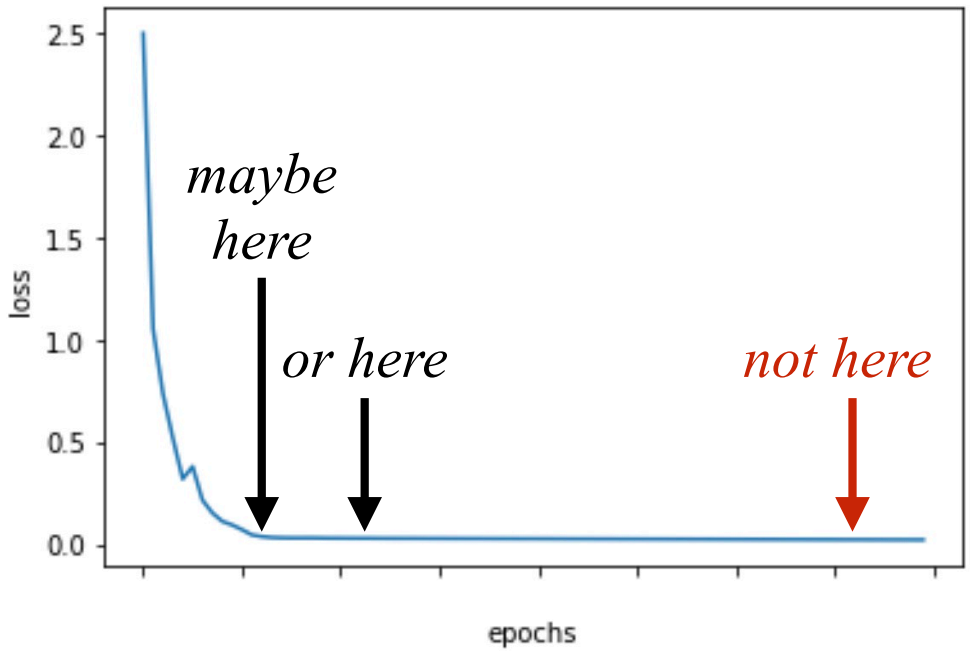
- create classifier in Keras
  - softmax activation function
  - categorical cross entropy for loss function
  - use no validation patterns
  - use L2 regularization to help avoid overfitting
    - *when you add a layer, include:*

```
kernel_regularizer=tf.keras.regularizers.l2(0.001)
```

train classifier for this fold

- create classifier in Keras
  - softmax activation function
  - categorical cross entropy for loss function
  - use no validation patterns
  - use L2 regularization to help avoid overfitting

- train classifier
  - pick a reasonable number of training epochs



train classifier for this fold

- create classifier in Keras
  - softmax activation function
  - categorical cross entropy for loss function
  - use no validation patterns
  - use L2 regularization to help avoid overfitting

- train classifier
  - pick a reasonable number of training epochs
  - train with **`fs_train_patterns`** and **`train_targets`** (see **`Homework8.ipynb`**)

test classifier for this fold

- test classifier
  - using **`network.evaluate()`** method for overall performance
  - using **`network.predict()`** method to get classification of individual test items
  - test with **`fs_test_patterns`** and **`test_targets`** (see **`Homework8.ipynb`**)

save classifier performance

- store in whatever data structure you've set up

report average of classifiers

- average across the 12 folds

testing for statistical significant?

remember, chance performance (1/8) is 12.5%

is 20% classification accuracy greater than chance?
what about 50%
what about 16%

in practice, you could have, for example, chance at 33% and see *statistically significant* classification at 36%

with "data science methods" like this, you often cannot rely on some standard (parametric) statistical test

permutation test

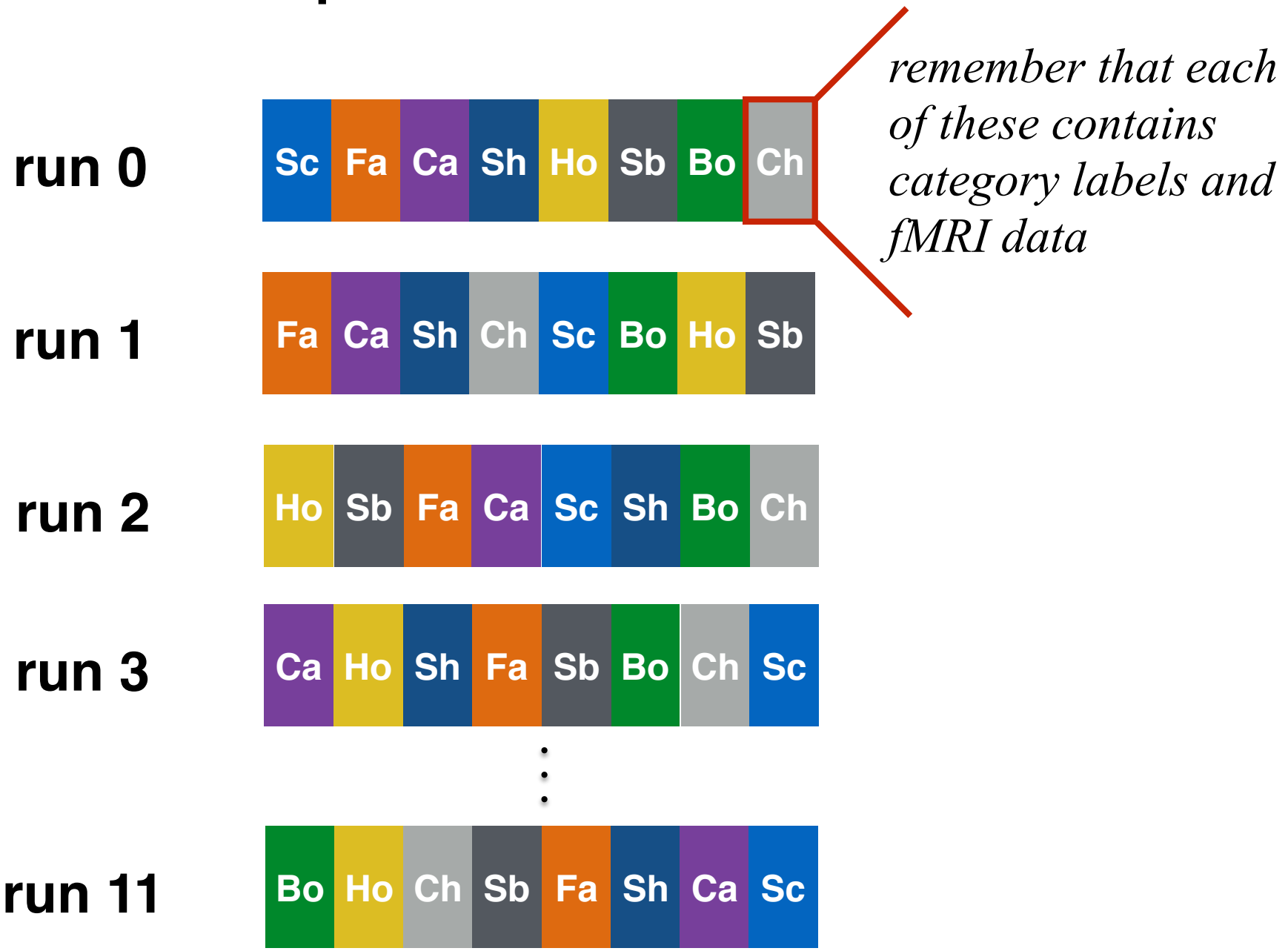
we're asking if we can predict the object category (label) from patterns of fMRI voxel activity

of course, when doing that, the label (chair, face, etc.) is consistently associated with measures of brain activity (fMRI) when the person is looking at that kind of object

run 3

permutation creates what you might expect "by chance"

permutation test



permutation test

what if we were to scramble the labels associated with different brain scans (don't touch brain scans)?

e.g., on one run, voxel measures when a face is shown would be labeled as being trials when a chair was shown, and on another run, voxel measures when a face is shown would be labeled as being trials when a shoe was shown

the performance for a classifier trained on that random shuffling of labels would indicate the level of performance you would expect for a classifier "by chance"

permutation test

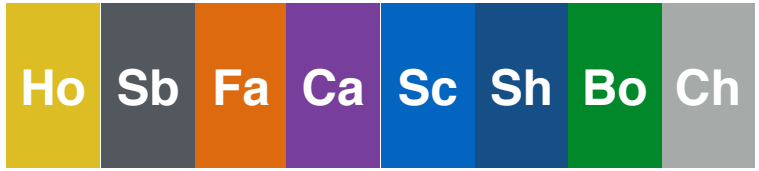
assume the colors represent the fMRI scans and letter represent labels

run 0

permute the labels

run 1

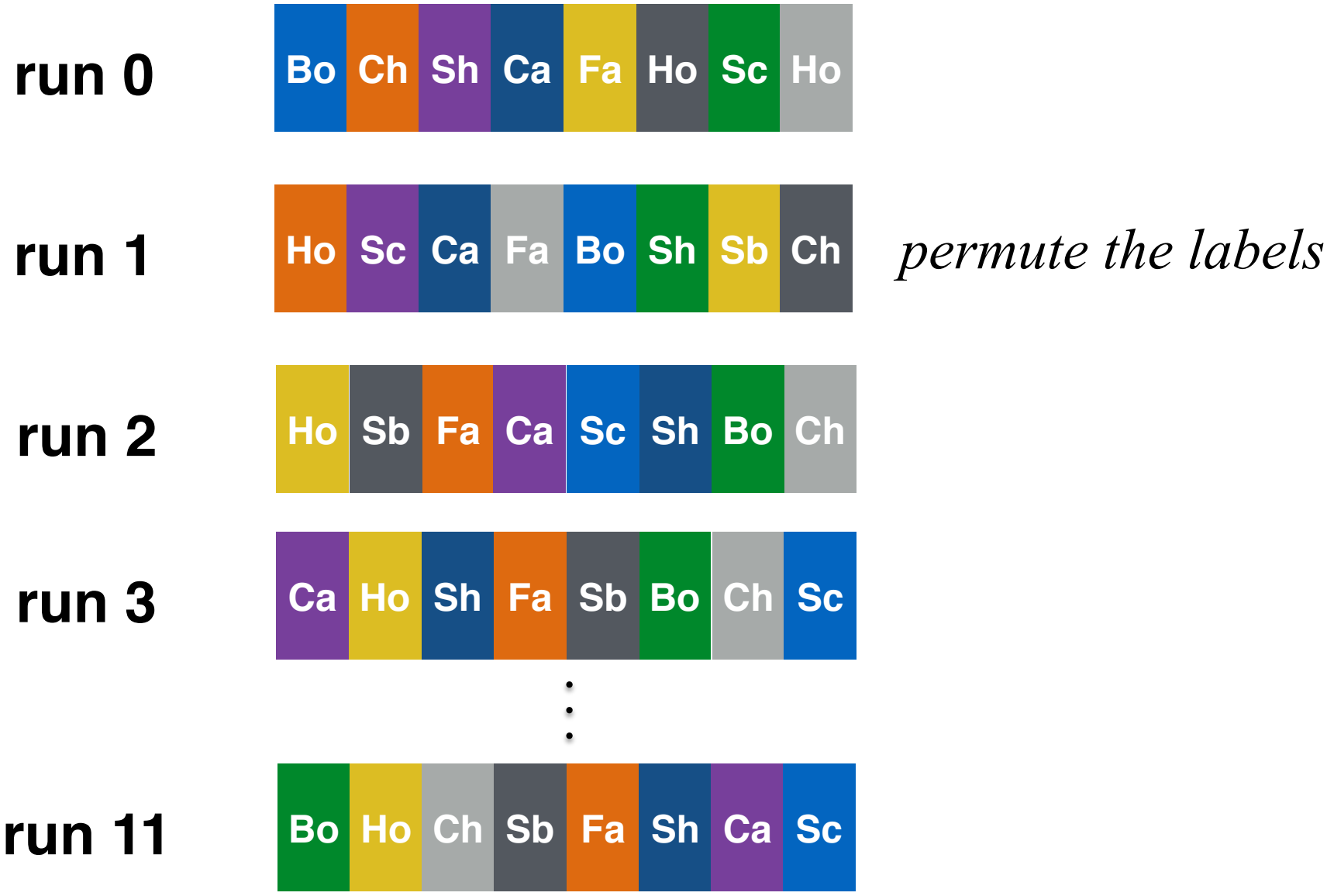
run 2



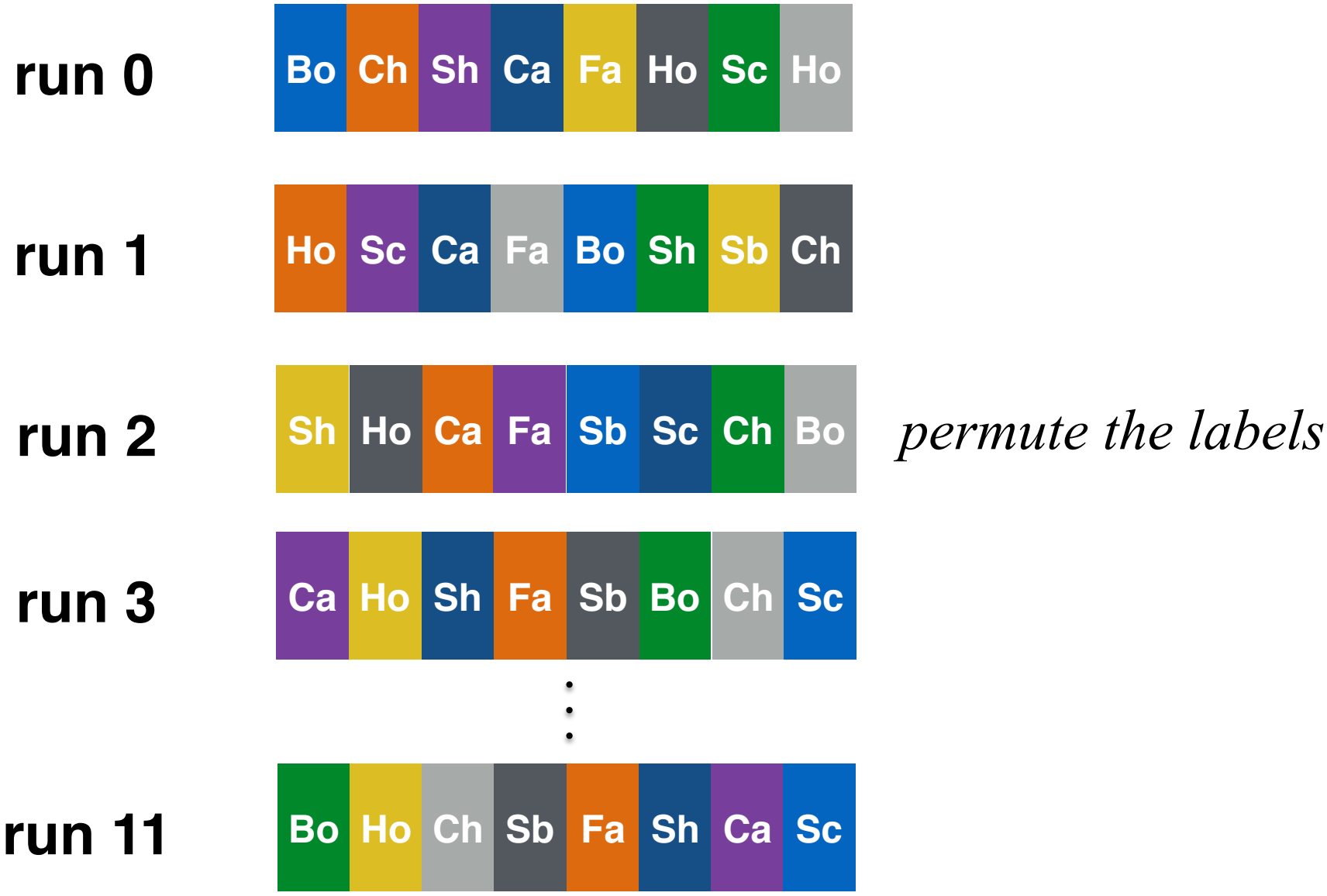
run 3

run 11

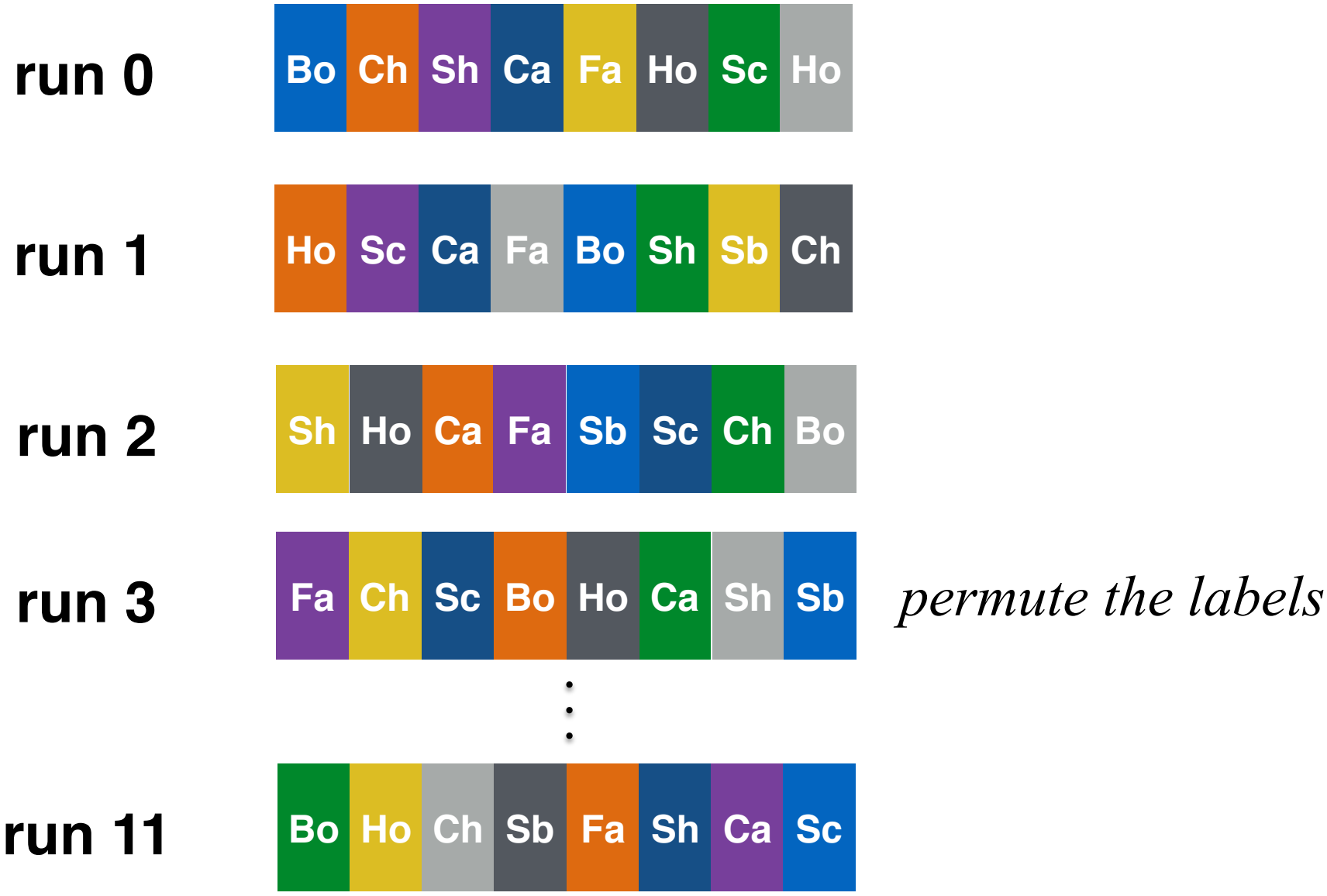
permutation test



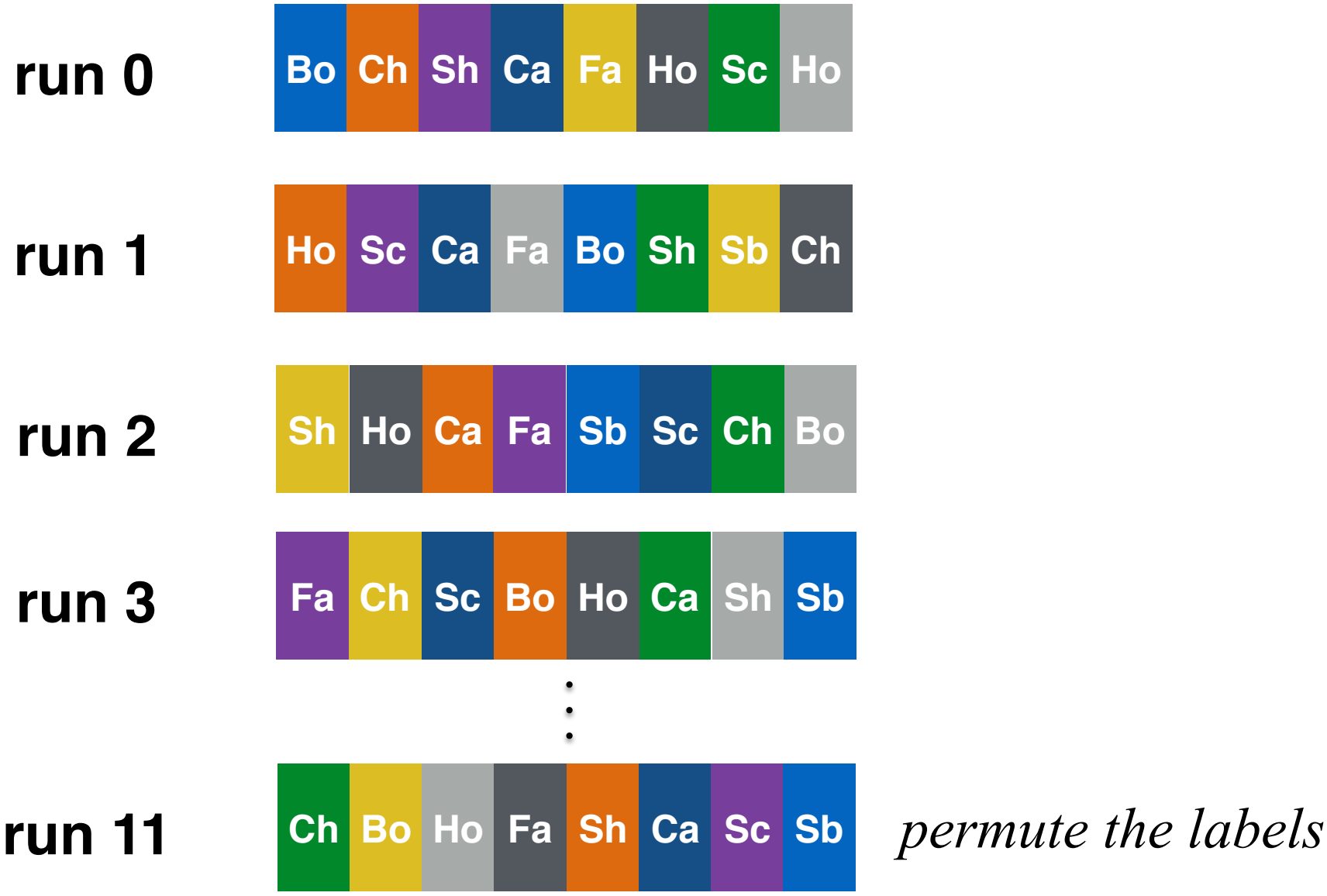
permutation test



permutation test



permutation test



permutation test

```
for p in range(n_perms):
```

in practice, `n_perms` = 5000

for Homework, `n_perms` = 10

(but should work for any values of `n_perms`*)*

permute labels

set up data structures to store classifier performance

```
for i in range(n_iters):
    for r in range(n_runs):
```

set up the fold r

do feature selection for this fold

train linear classifier for this fold

test linear classifier for this fold

save classifier performance

report average of classifiers

save average performance to array to calculate chance range

permutation test

```
for p in range(n_perms):
```

permute labels

gave you a function to do this in `Homework8.ipynb`

set up data structures to store classifier performance

```
for i in range(n_iters):
    for r in range(n_runs):
```

set up the fold r

do feature selection for this fold

train linear classifier for this fold

test linear classifier for this fold

save classifier performance

report average of classifiers

save average performance to array to calculate chance range

permutation test

```
for p in range(n_perms):
```

permute labels

set up data structures to store classifier performance

```
for i in range(n_iters):
    for r in range(n_runs):
```

set up the fold r

do feature selection for this fold

train linear classifier for this fold

test linear classifier for this fold

save classifier performance

report average of classifiers

save average performance to array to calculate chance range

need to do this part with the permuted labels

permutation test

```
for p in range(n_perms):
```

permute labels

set up data structures to store classifier performance

```
for i in range(n_iters):
    for r in range(n_runs):
```

set up the fold r

do feature selection for this fold

train linear classifier for this fold

test linear classifier for this fold

save classifier performance

report average of classifiers

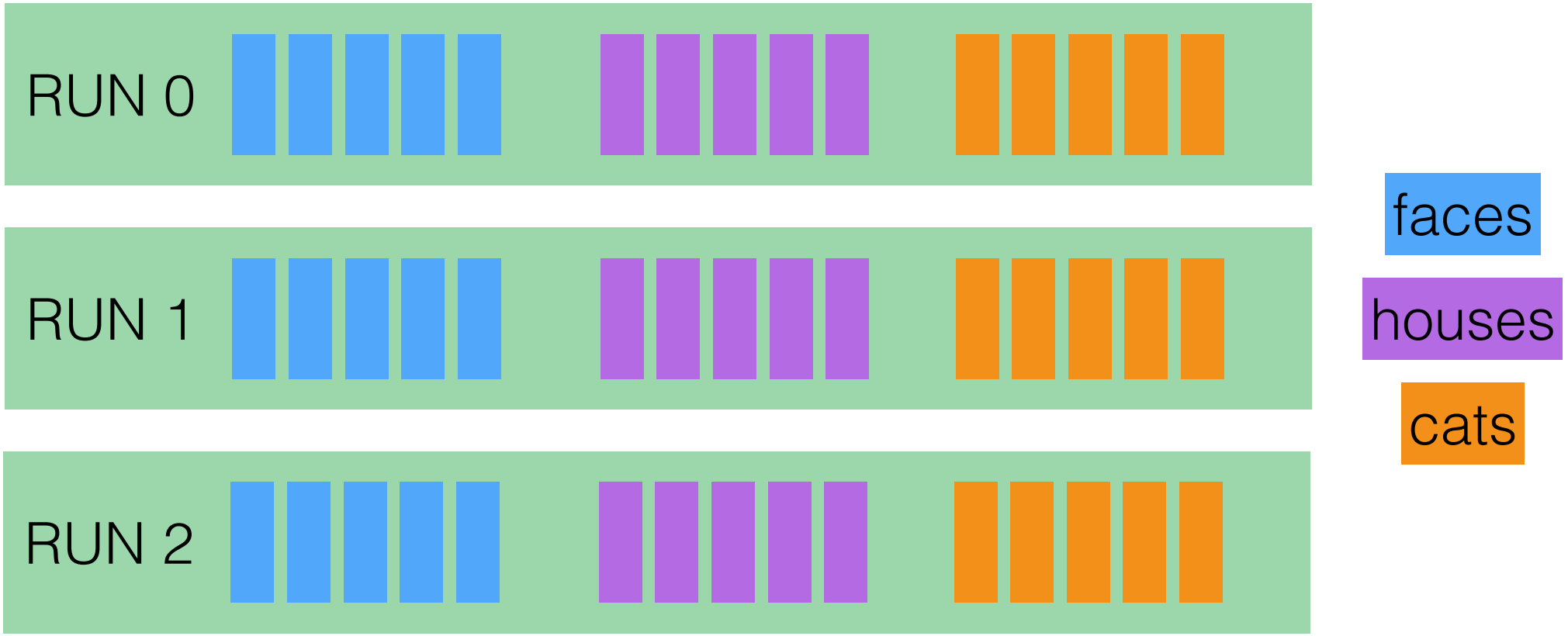
save average performance to array to calculate chance range

array of `n_perms` *values of classifier performance for permuted labels*

permutation test

with true labels

classification accuracy = 45%

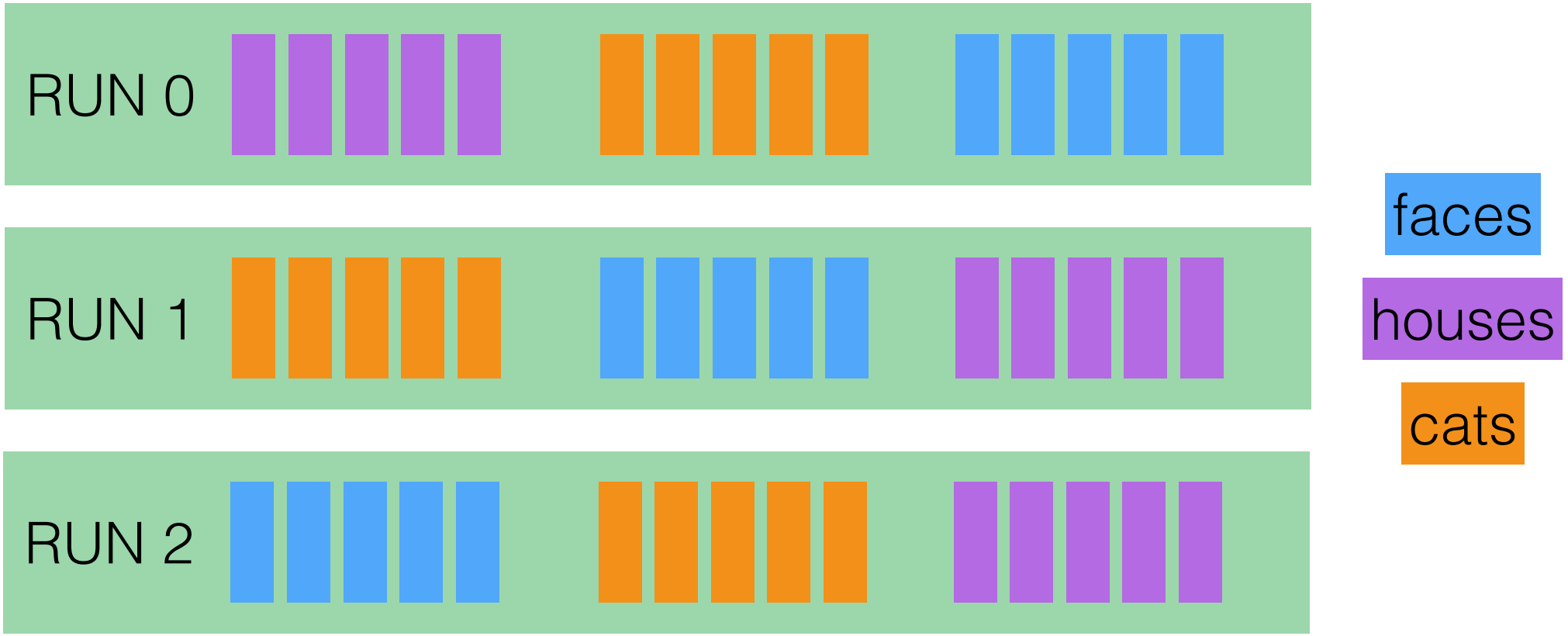


average performance across all runs = “observed accuracy”

permutation test

with scrambled labels

classification accuracy = 14%

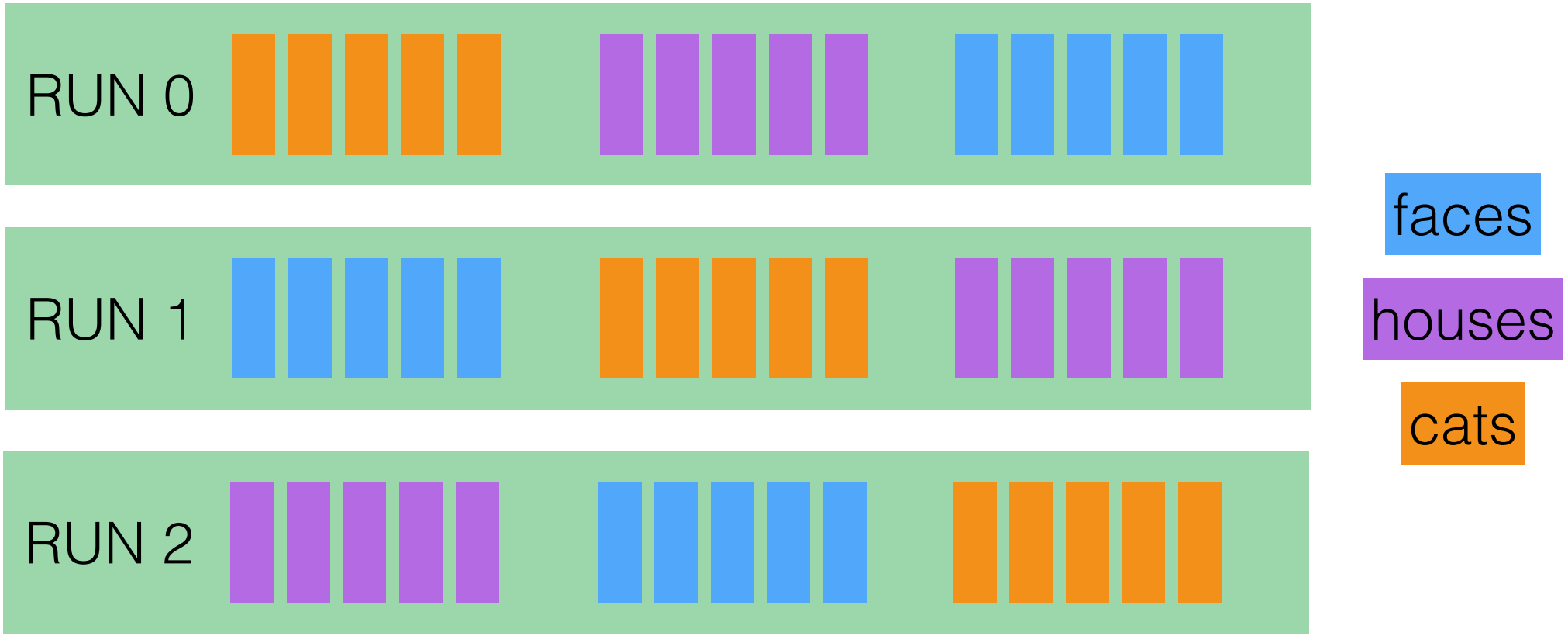


average performance across all runs = “scrambled accuracy #1”

permutation test

with scrambled labels

classification accuracy = 12%



average performance across all runs = “scrambled accuracy #2”

permutation test

observed accuracy = 45%

scrambled accuracy = [14.0,

scrambled accuracy #1

permutation test

observed accuracy = 45%

scrambled accuracy = [14.0, 12.0,

scrambled accuracy #2

permutation test

observed accuracy = 45%

scrambled accuracy = [14.0, 12.0, 8.5,

scrambled accuracy #2

permutation test

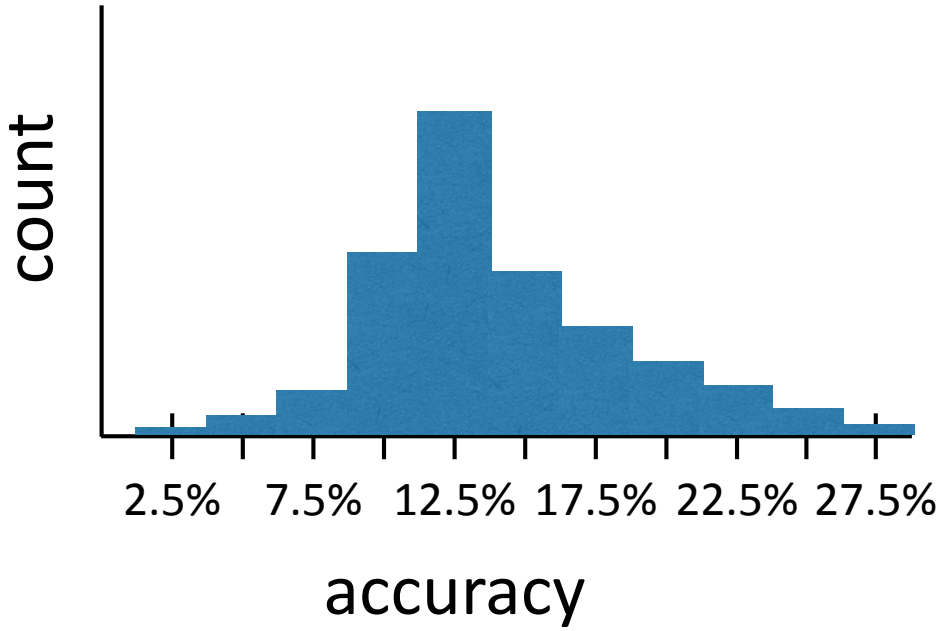
observed accuracy = 45%

scrambled accuracy = [14.0, 12.0, 8.5, 14.3, 8.3, 17.4, 12.9, 11.1, ...

permutation test

observed accuracy = 45%

scrambled accuracy = [14.0, 12.0, 8.5, 14.3, 8.3, 17.4, 12.9, 11.1, ...

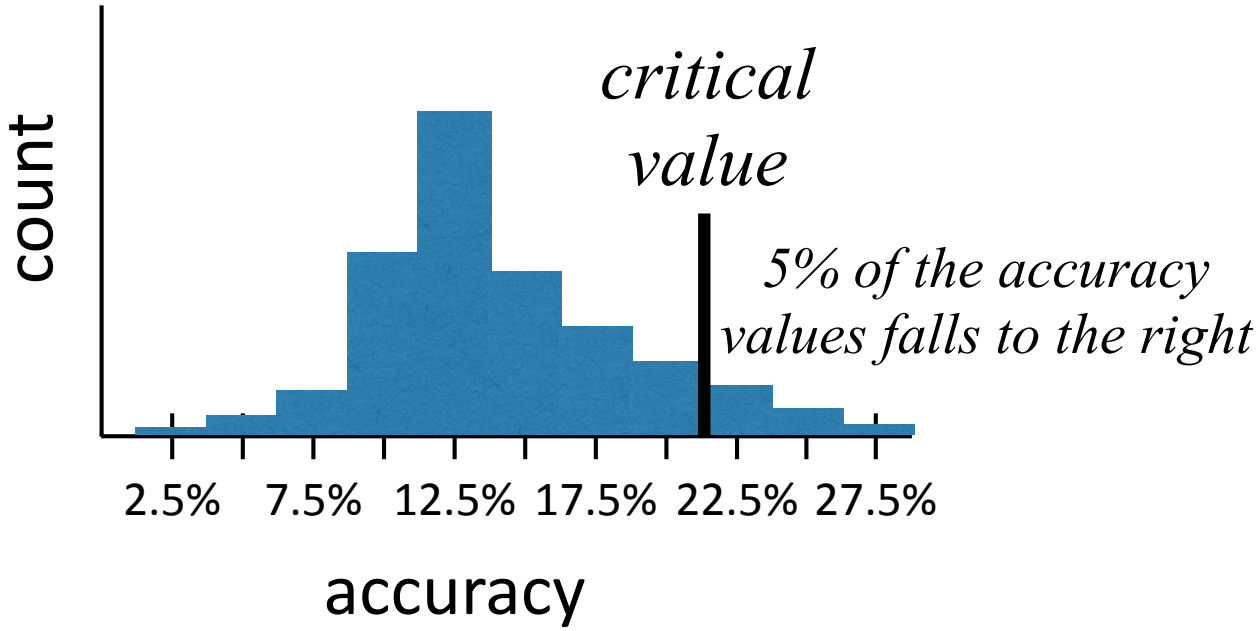


histogram of "chance" performance across 1000 permuted replications of the data

permutation test

observed accuracy = 45%

scrambled accuracy = [14.0, 12.0, 8.5, 14.3, 8.3, 17.4, 12.9, 11.1, ...

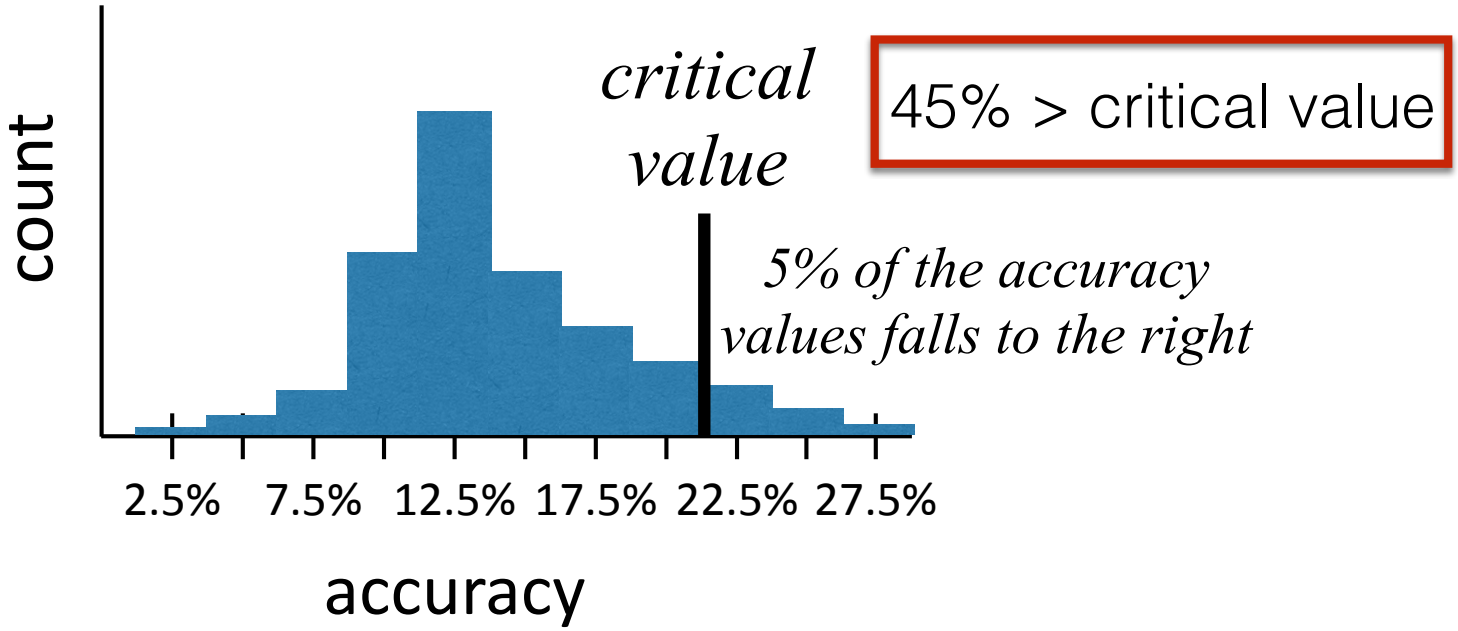


histogram of "chance" performance across 1000 permuted replications of the data

permutation test

observed accuracy = 45%

scrambled accuracy = [14.0, 12.0, 8.5, 14.3, 8.3, 17.4, 12.9, 11.1, ...



histogram of "chance" performance across 1000 permuted replications of the data