Advanced Cognitive Neuroscience

Week 41: Multivariate statistics

The course plan

Week 36:
Lesson 0: What is it all about?
Class 0: Setting up UCloud and installing MNE-Python

Week 37:
No Teaching

Week 38:
Lesson 1: Workshop paradigm: Measuring visual subjective experience + MR Recordings
Class 1: Running an MEG analysis of visual responses

Week 39:
MEG workshop: Measuring and predicting visual subjective experience

Week 40:
Lesson 2: Basic physiology and Evoked responses
Class 2: Evoked responses to different levels of subjective experience

Week 41:
Lesson 3: Multivariate statistics
Class 3: Predicting subjective experience in sensor space
Deadline for feedback: Video Explainer

Week 42:
Autumn Break

Week 43:
Lesson 4: Forward modelling and dipole estimation
Class 4: Creating a forward model

Week 44:
Lesson 5: Inverse modelling: Minimum-norm estimate
Class 5: Predicting subjective experience in source space

Week 45:
Lesson 6: Inverse modelling: Beamforming
Class 6: Predicting subjective experience in source space, continued

Week 46:
Lesson 7: What about that other cortex? - the cerebellar one
Class 7: Oral presentations (part 1)
Deadline for feedback: Lab report

Week 47:
Lesson 8: Guest lecture: Laura Bock Paulsen: Respiratory analyses
Class 8: Oral presentations (part 2)

Week 48:
Lesson 9: Guest lecture: Barbara Pomiechowska: Using OPM-MEG to study brain and cognitive development in infancy
Class 9: Oral presentations (part 3)

Week 49:
Lesson 0 again: What was it all about?
Class 10: Oral presentations (part 4)

Video explainer

- Concepts to explain
  - post-synaptic potentials
  - current dipole,
  - open field vs closed field
  - radial and tangential sources
  - volume conduction
  - evoked responses
- Deadline Thursday **this** week (for feedback)

Video explainer

Just you talking, explaining the relevant concepts. You can gesture, but no props and no screen recordings (only I will see it)

They’re already coming in – I am looking very much forward to seeing them all

Learning goals

- You should (re)learn:
  - the basics of logistic regression
  - what targets could be aimed at in our experiment
  - what the pros and cons of multivariate statistics are for MEG data
    - what are the differences between sensor space decoding and source space decoding
- How to set up a pipeline
  - Standardisation
  - Feature selection (e.g. PCA)
  - Finding optimal hyperparameters (e.g. Grid Search)

Let’s have a look at an iris flower

se·pal (sē′pəl)

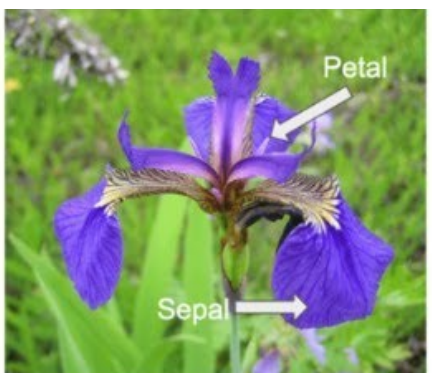
n.

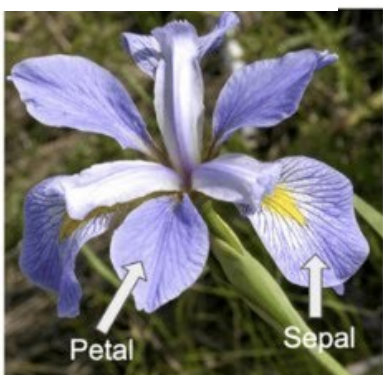
One of the usually green leaflike structures composing the outermost part of a flower. Sepals often enclose and protect the bud and may remain after the fruit forms.

pet·al (pĕt′l)

n.

One of the often brightly colored parts of a flower immediately surrounding the reproductive organs; a division of the corolla.

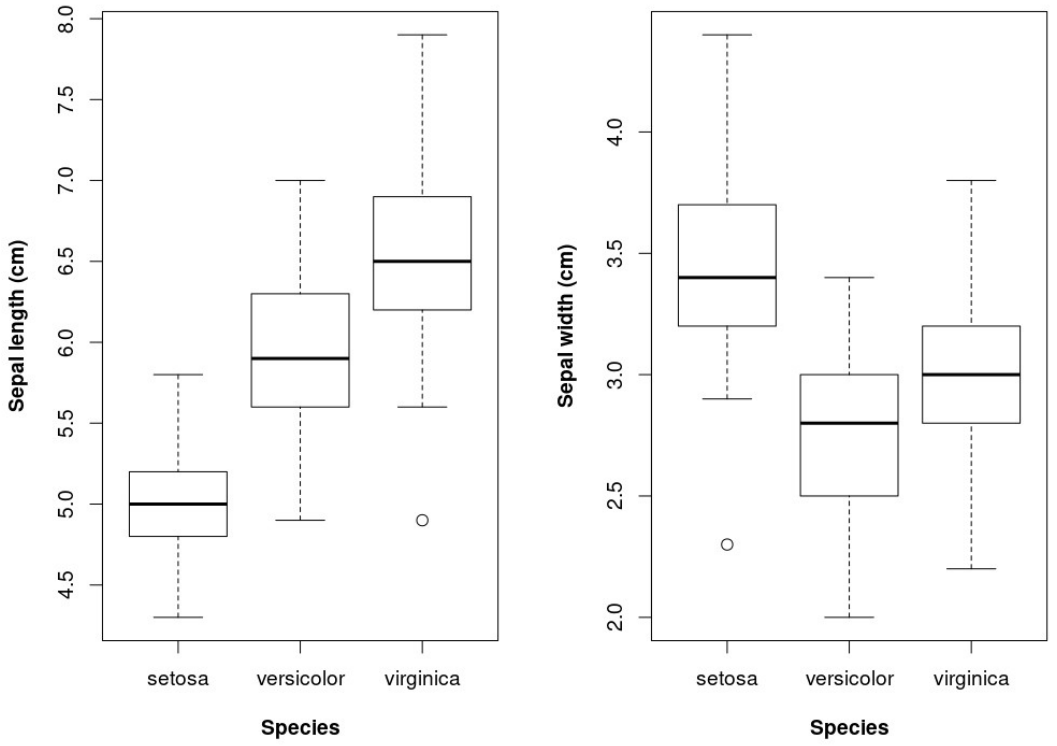




https://i.ytimg.com/vi/E8Nrl9QLNq8/maxresdefault.jpg

Encoding

Encoding: *describing* the *category* based on the *measurement* variable



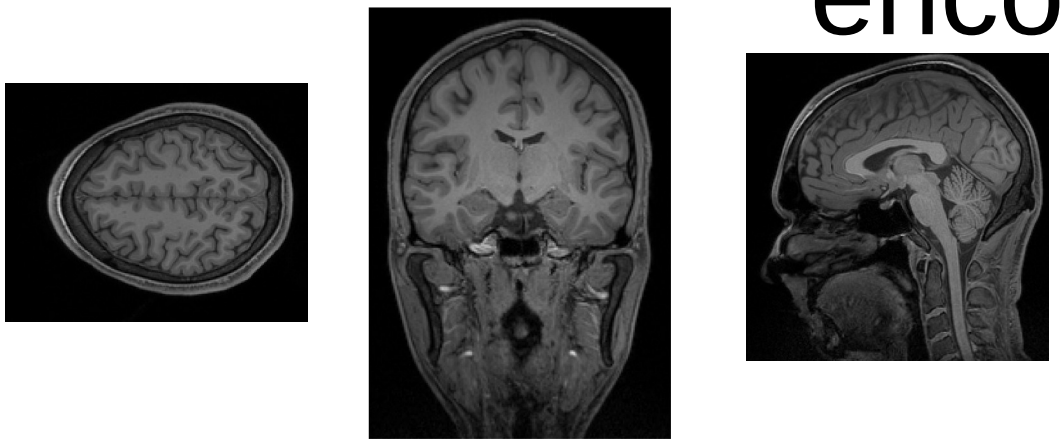
Encoding

```
> print(anova(lm(Sepal.Length ~ Species, data=iris)))
Analysis of Variance Table

Response: Sepal.Length
           Df Sum Sq Mean Sq F value    Pr(>F)
Species     2 63.212  31.606  119.26 < 2.2e-16 ***
Residuals 147 38.956   0.265
```

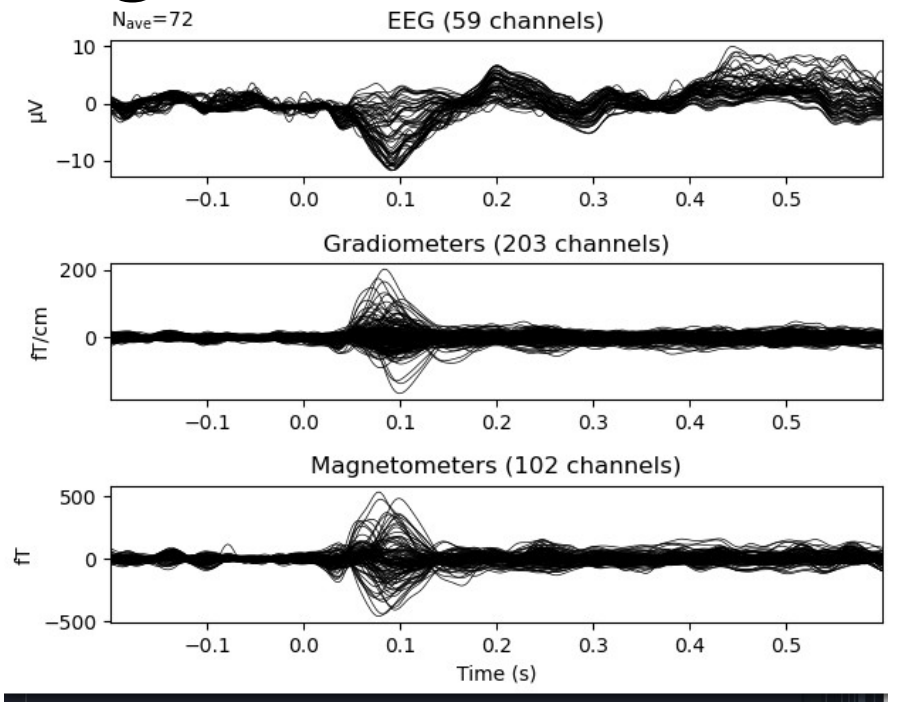
```
> print(anova(lm(Sepal.Width ~ Species, data=iris)))
Analysis of Variance Table

Response: Sepal.Width
           Df Sum Sq Mean Sq F value    Pr(>F)
Species     2 11.345  5.6725   49.16 < 2.2e-16 ***
Residuals 147 16.962  0.1154
```

Encoding: *statistics* can then help you make *inferences* as to whether *measurements* differ between the *categories,* i.e. whether *categories* capture meaningful differences

Richness of neuroimaging data encoding



(f)MRI typically contains in the order of 10-20 million voxels



An MEG dataset will have ~300,000 data points per second

Family-wise error rate

The probability that you will commit at least one Type I error, i.e. rejecting the null hypothesis when the null hypothesis is actually true, also called a *false positive*

This increases with the number of tests you make. For encoding approaches to whole brain data, this probability is practically 1.

Decoding with **logistic regression**

At least four ingredients needed

GENERALISED LINEAR MODEL

1) A data vector: $y = (y_1, ..., y_n)$

2) Predictors: X and coefficients β, forming a linear predictor $X\beta$

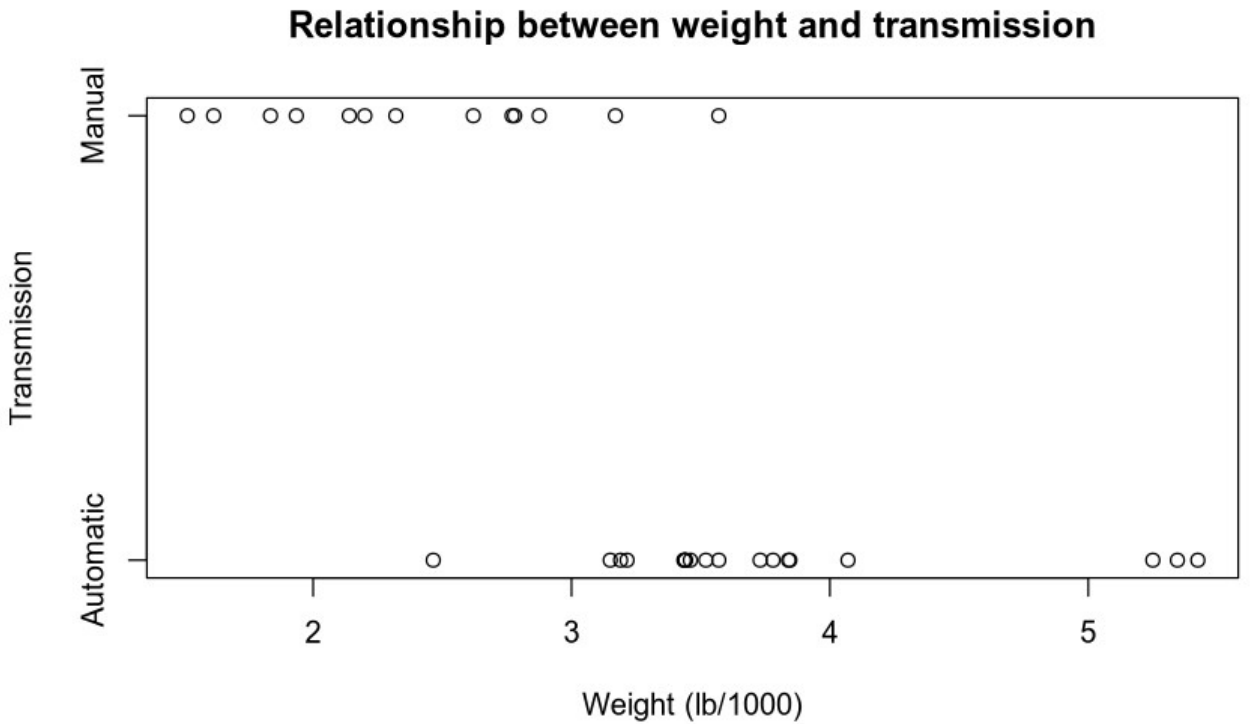
3) A *link function* g: yielding a vector of transformed data $\hat{y} = g^{-1}(X\hat{\beta})$ that are used to model the data

4) A data distribution: $p(y|\hat{y})$

$(X\beta = \beta_0 + X_1\beta_1 + ... + X_k\beta_k)$

Gelman A, Hill J (2006) Data Analysis Using Regression and Multilevel/Hierarchical Models. Cambridge University Press: Chapter 6

1) A data vector : $y = (y_1, \ldots, y_n)$



or relation between MEG:
b(*t*) or ***s***(***r***, *t*)

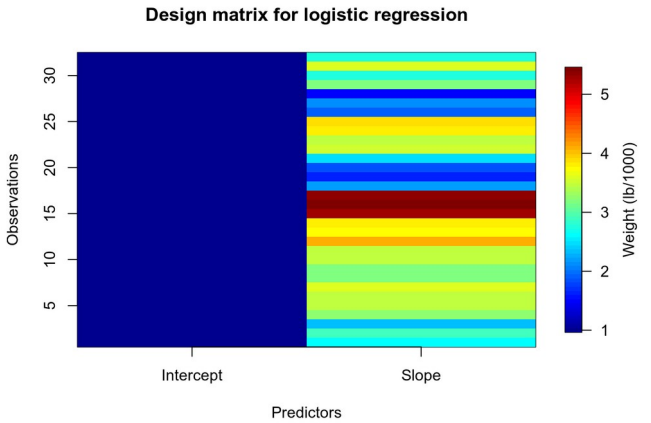
and subjective experience category:

NE; WG; ACE; CE

2) Predictors: X and coefficients β, forming a linear predictor $X\beta$

```r
X <- matrix(nrow=n, ncol=2) ## choosing two predictors
X[, 1] <- 1 ## modelling an intercept
X[, 2] <- mtcars$wt ## modelling a linear predictor based on weight
library(fields)
image.plot(x=1:dim(X)[2], y=1:dim(X)[1], z=t(X),
           xlab='Predictors', ylab='Observations', xaxt='n',
           main='Design matrix for logistic regression',
           legend.lab='Weight (lb/1000)')
axis(1, 1:dim(X)[2], c('Intercept', 'Slope'))
```

$X =$

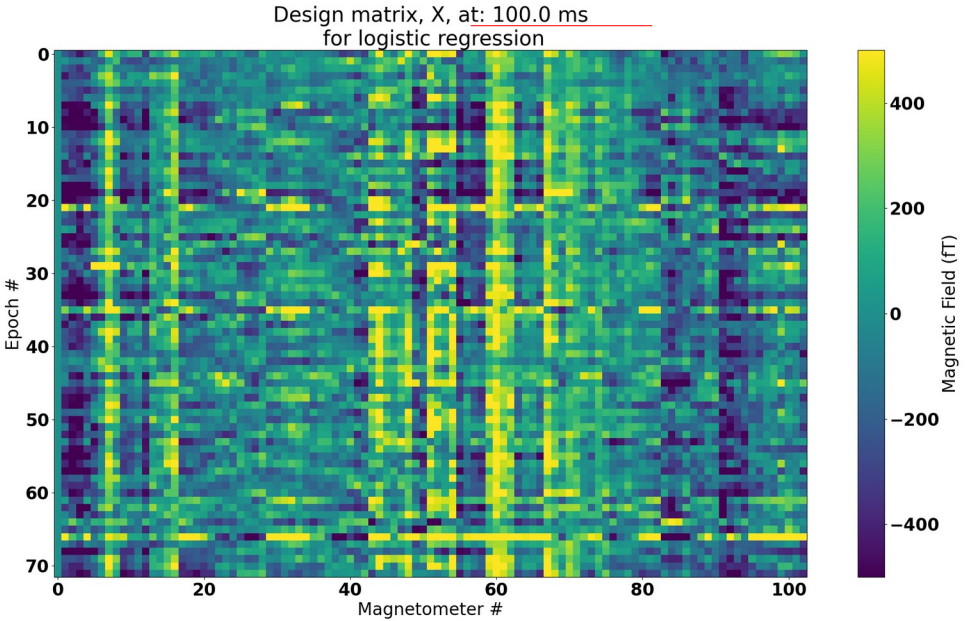


$\beta =$ to be estimated

2) Predictors: X and coefficients β, forming a linear predictor $X\beta$

```python
def plot_design_matrix_at_time_index(time_index):

    X = 1e15 * epochs_sample['LA'].get_data()[:, :, time_index]
    X = np.insert(X, 0, 1, axis=1) # adding intercept term

    plt.figure()
    plt.imshow(X, vmin=-500, vmax=500)
    plt.colorbar(label='Magnetic Field (fT)')
    plt.ylabel('Epoch #')
    plt.xlabel('Magnetometer #')
    plt.title(f'Design matrix, X, at: '
              f'{1e3*np.round(epochs_sample.times[time_index], 3)}'
              ' ms\n for logistic regression' )
    plt.show()

plot_design_matrix_at_time_index(180)
plot_design_matrix_at_time_index(30)
```

$X =$

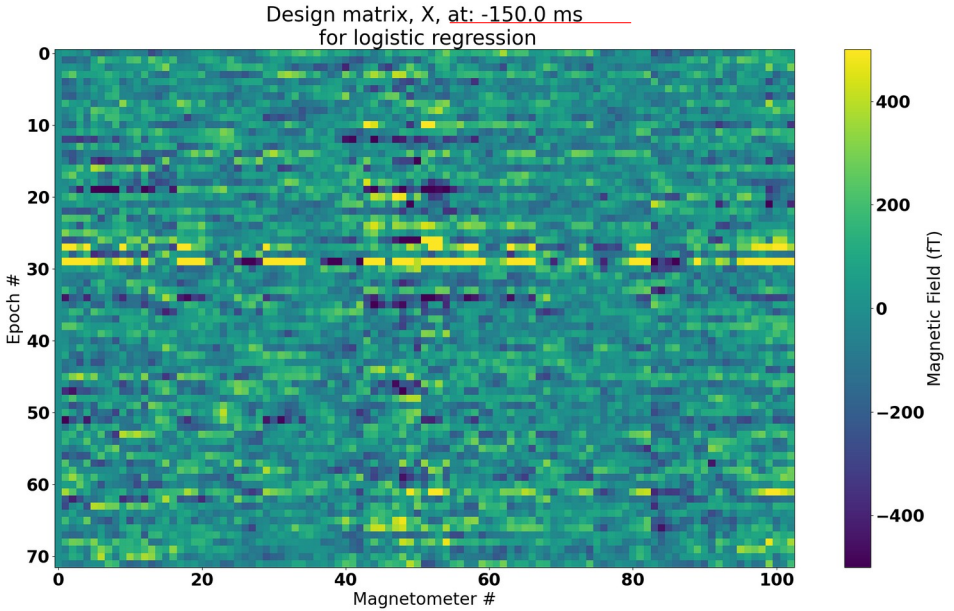


$\beta =$ to be estimated

2) Predictors: X and coefficients β, forming a linear predictor $X\beta$

```python
def plot_design_matrix_at_time_index(time_index):

    X = 1e15 * epochs_sample['LA'].get_data()[:, :, time_index]
    X = np.insert(X, 0, 1, axis=1) # adding intercept term

    plt.figure()
    plt.imshow(X, vmin=-500, vmax=500)
    plt.colorbar(label='Magnetic Field (fT)')
    plt.ylabel('Epoch #')
    plt.xlabel('Magnetometer #')
    plt.title(f'Design matrix, X, at: '
              f'{1e3*np.round(epochs_sample.times[time_index], 3)}'
              ' ms\n for logistic regression' )
    plt.show()

plot_design_matrix_at_time_index(180)
plot_design_matrix_at_time_index(30)
```

$X =$



$\beta =$ to be estimated

After fitting the model

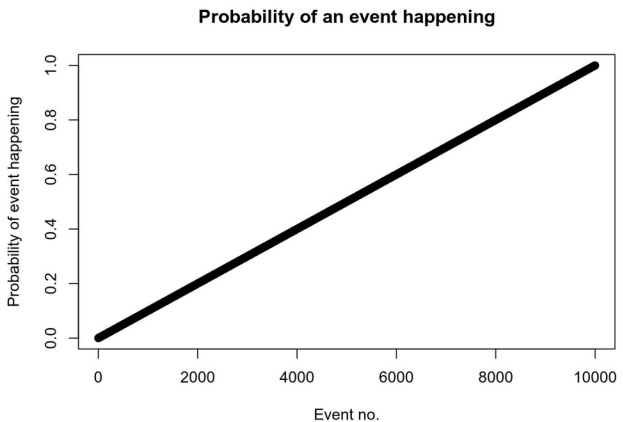
$\hat{\beta} =$

```
In [5]: logr.coef_
Out[5]:
array([[-0.00549997, -0.00247487,  0.00011406,  0.00248419, -0.01129127,
         0.00295924,  0.00260175, -0.01114961,  0.01091916,  0.00120471,
        -0.00276249,  0.00388734, -0.01138881,  0.00383627,  0.00463872,
         0.00194857,  0.00820409, -0.00615871,  0.00224653, -0.00145875,
        -0.0048058 ,  0.01038711,  0.00713602, -0.00214013, -0.00136313,
        -0.00425   , -0.00282781, -0.00192131,  0.0053184 ,  0.00381331,
         0.0056061 , -0.00179183,  0.01010728,  0.00087808,  0.0025017 ,
         0.00970538, -0.0009978 , -0.0010561 ,  0.00575529,  0.00017711,
         0.0005281 ,  0.00401336, -0.00047705, -0.00200831,  0.00364203,
         0.00103059,  0.00265459,  0.00472789,  0.00357972,  0.00324831,
        -0.00582481,  0.00646803,  0.00024858,  0.00310502, -0.00904192,
         0.00329844,  0.00449159,  0.00576189, -0.00551431,  0.01008898,
         0.00772441,  0.00558567, -0.00814402,  0.00061284, -0.00207037,
        -0.00637707, -0.00559792,  0.00502742, -0.00451853,  0.00105205,
        -0.0053851 ,  0.00140829, -0.00181343,  0.00484863, -0.0004228 ,
        -0.00778388, -0.00057924,  0.00637361, -0.0013187 , -0.00433823,
         0.00119534,  0.00856528, -0.00397713, -0.00025439,  0.00135553,
         0.00229648, -0.00373167]])
```

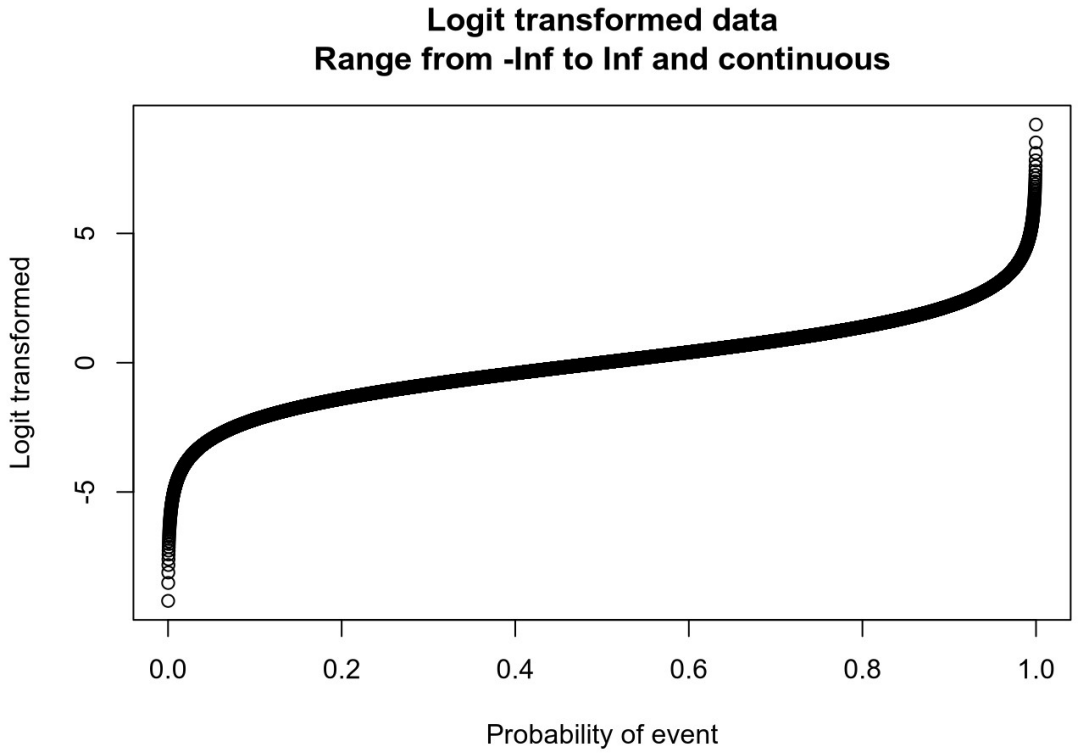
3) A link function g: yielding a vector of transformed data $\hat{y} = g^{-1}(X\hat{\beta})$ that are used to model the data

```r
g     <- function(x) log(x / (1 - x)) ## logit
inv.g <- function(x) exp(x) / (1 + exp(x)) ## logit⁻¹
```

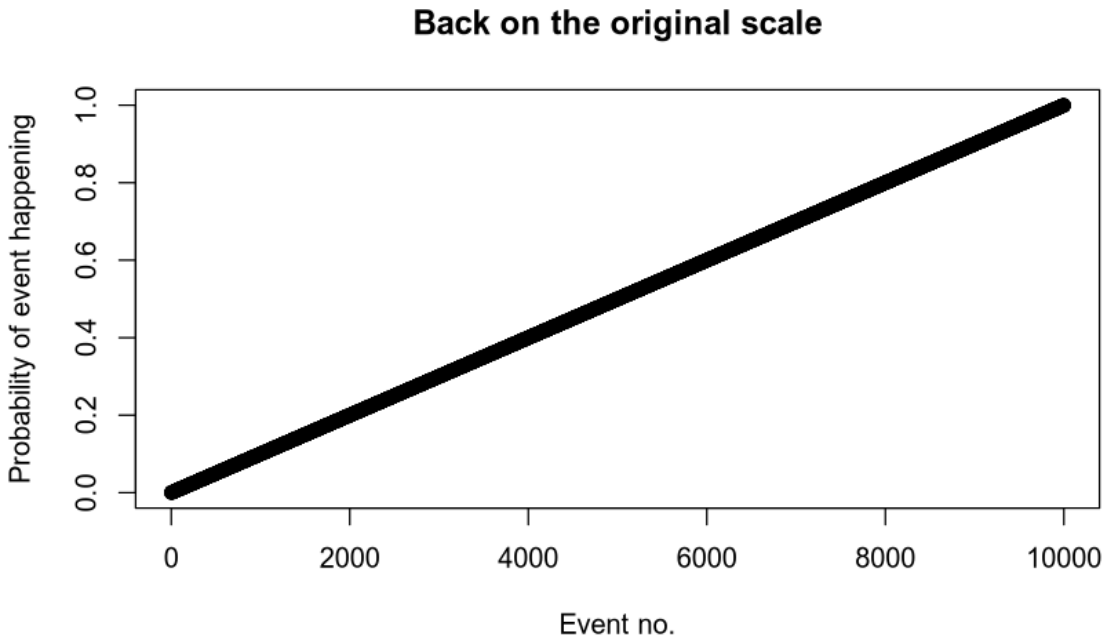
```r
y <- seq(0.0001, 0.9999, 0.0001)
x <- 1:length(y)
plot(x, y, main='Probability of an event happening', xlab='Event no.',
     ylab='Probability of event happening')
```



```
plot(y, g(y), xlab='Probability of event', ylab='Logit transformed',
     main='Logit transformed data\nRange from -Inf to Inf and continuous')
```



```
plot(x, inv.g(g(y)), main='Back on the original scale',
     xlab='Event no.', ylab='Probability of event happening')
```

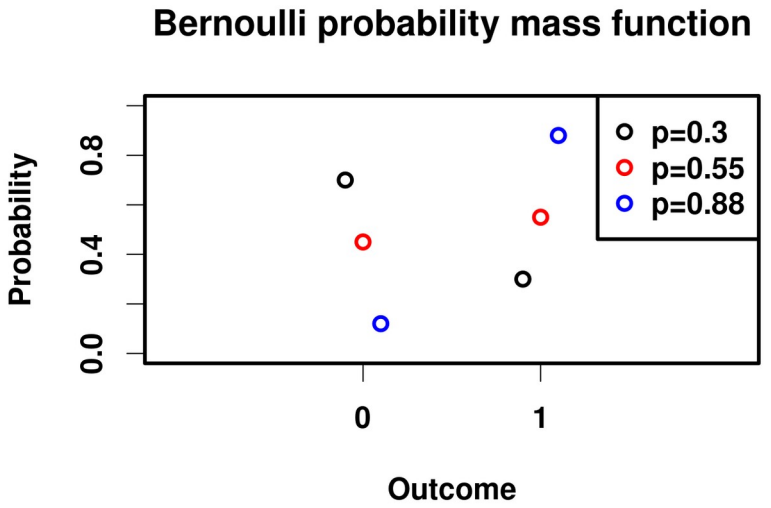


4) A data distribution: $p(y|\hat{y})$

$$\text{PMF}_{\text{Bernouilli}} = p^k(1-p)^{1-k} = \begin{cases} p & \text{if } k=1, \\ q=1-p & \text{if } k=0. \end{cases}$$

$$p \in [0,1]; k \in \{0,1\}$$

```
dbernoul <- function(p, k) p^k * (1 - p)^(1 - k)

x <- c(0, 1)
pmf <- dbernoul(p=0.3, x)
par(font.lab=2, font.axis=2)
plot(x -0.1, pmf, xaxt='n', xlab='Outcome', ylab='Probability', ylim=c(0,
1),
     xlim=c(-1.1, 2.1), main='Bernoulli probability mass function')
axis(side=1, at=c(0, 1), labels=c(0, 1))

pmf <- dbernoul(p=0.55, x)
points(x, pmf, col='red')

pmf <- dbernoul(p=0.88, x)
points(x + 0.1, pmf, col='blue')

legend('topright', pch=1, col=c('black', 'red', 'blue'),
       legend=c('p=0.3', 'p=0.55', 'p=0.88'), text.font=2)
```



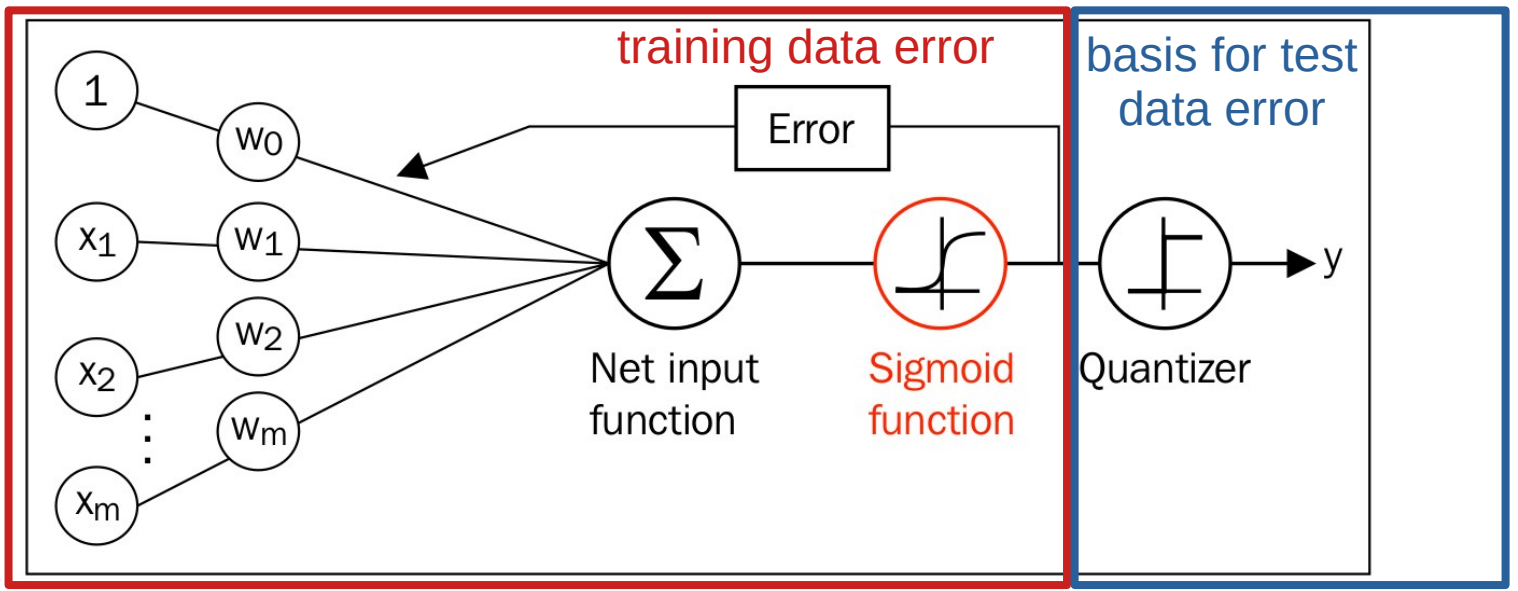
In our case: 0 and 1 would be the categories we are decoding

WHY LOGISTIC REGRESSION?

Together with a quantiser function, we can make categorical predictions

Recap – logistic regression

adapted from (p. 58: Raschka, 2015)



Features (known) ***MEG***

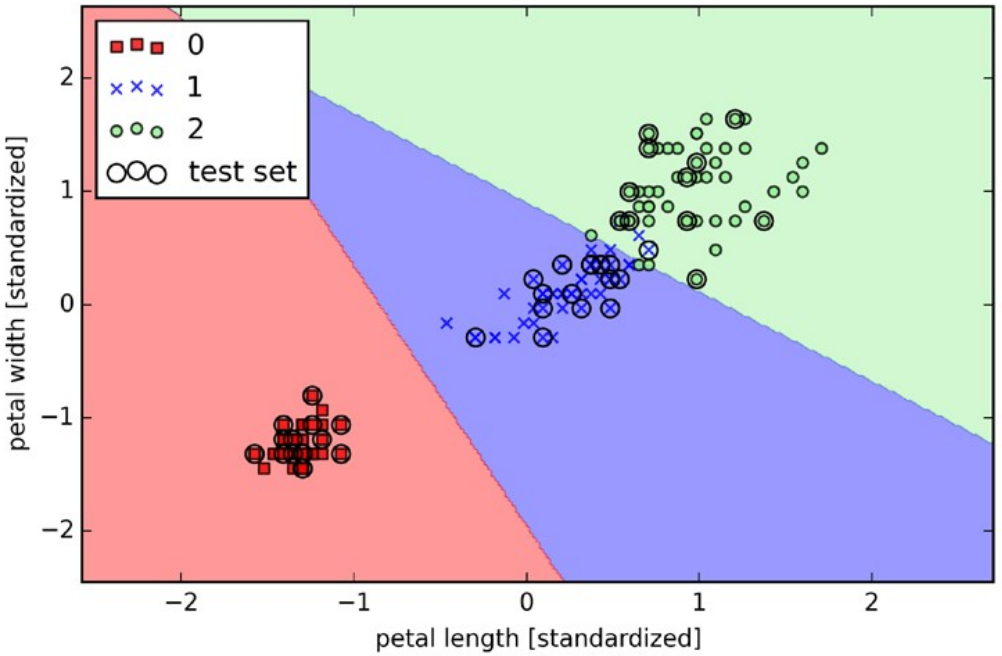
Weights (unknown)

Outcome (known) ***Subj. rating***

WANTED: an algorithm that converges with linearly separable regions that minimise the number of errors made

Something like this

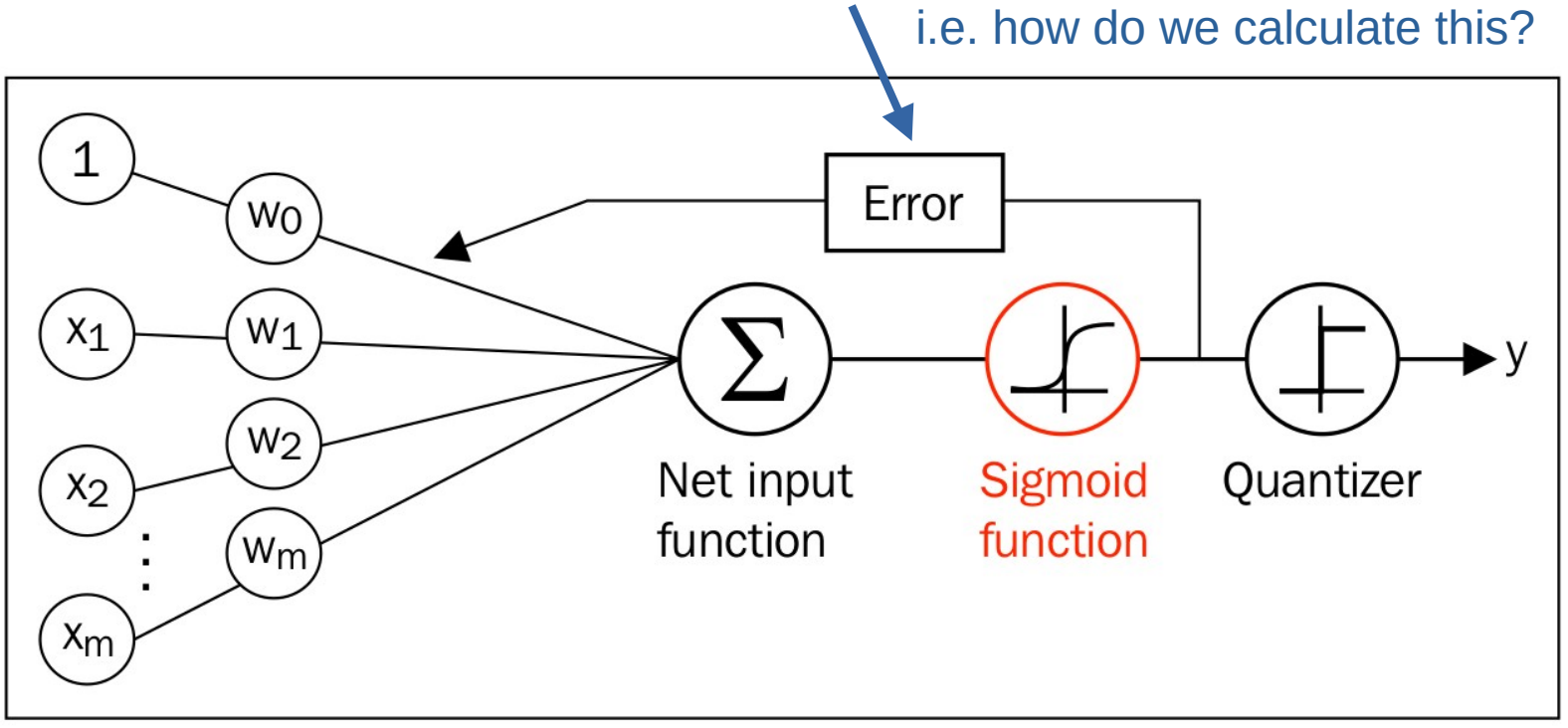
LOGISTIC REGRESSION



(p. 63: Raschka, 2015)

Separates flowers, while keeping errors at a minimum

What's our cost function, $J(\boldsymbol{w})$?

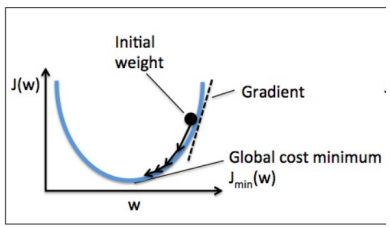


Updating weights

$$\Delta \boldsymbol{w} = -\eta \nabla J(\boldsymbol{w})$$

$\Delta \boldsymbol{w}$: change in $\boldsymbol{w}$
η: learning rate
$\nabla J(\boldsymbol{w})$: gradient of cost function $J(\boldsymbol{w})$

$$J(\boldsymbol{w})_{LOGISTIC} = -\sum_{i=1}^{n} \boldsymbol{y}_i \log(\hat{\boldsymbol{y}}_i) + (1-\boldsymbol{y}_i)\log(1-\hat{\boldsymbol{y}}_i)$$



(p. 40: Raschka, 2015)

```python
def sigmoid(z):
    return np.exp(z) / (1 + np.exp(z))
    # return 1 / (1 + np.exp(-z))

def cost_function(y, yhat):
    return -np.sum(y * np.log(yhat) + (1 - y) * np.log(1 - yhat))

def gradient(X, y, yhat):
    return np.dot((y - yhat), X)

class LogisticRegressionGD:
    def __init__(self, eta=0.0001, n_iterations=1000, tol=1e-4):
        self.eta = eta
        self.n_iterations = n_iterations
        self.w_ = None
        self.cost_ = []
        self.tol = tol
        self.n_iter_ = 0
```

```python
def fit(self, X, y):

    X = np.insert(X, 0, 1, axis=1) # adding intercept term
    self.w_ = np.zeros(X.shape[1])
    # Gradient Descent
    for _ in range(self.n_iterations):
        self.n_iter_ += 1 ## go through number of iterations
        # Calculate linear combination of features and weights
        self.linear_model = np.dot(X, self.w_)
        # Apply sigmoid function to get probabilities
        self.yhat = sigmoid(self.linear_model)

        self.cost_.append(cost_function(y, self.yhat))

        # Update weights using the gradient
        self.w_ += self.eta * gradient(X, y, self.yhat)
        ## stop when we have found a minimum
        if self.n_iter_ > 1: ## we need at least two observations
            diff = abs(self.cost_[-1] - self.cost_[-2])
            if diff < self.tol:
                break
```

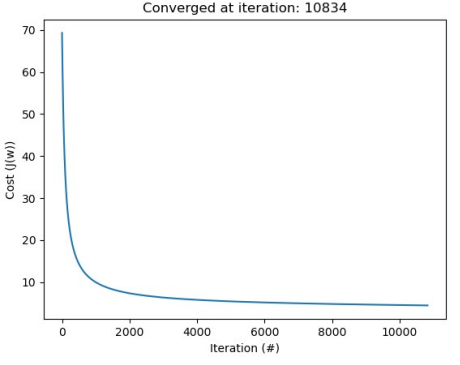
```python
X, y = load_iris(return_X_y=True)
X = X[y < 2, 0:2]  ## petal length and sepal length
y = y[y < 2] ## two flowers only, 0 and 1

y_train = y[10:90]
X_train = X[10:90, :]

y_test = np.concatenate((y[0:10], y[90:100]))
X_test = np.concatenate((X[0:10, :], X[90:100, :]))
```

```python
def predict(self, X):

    X = np.insert(X, 0, 1, axis=1) # adding intercept term
    yhat = np.dot(X, self.w_)
    self.y_predict = (yhat >= 0.50).astype(int)
```

```python
logreg = LogisticRegressionGD(eta=0.001, n_iterations=100000, tol=1e-4)
logreg.fit(X_train, y_train)
logreg.predict(X_test)
```

```
In [143]: print(logreg.y_predict)
[0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1]
```

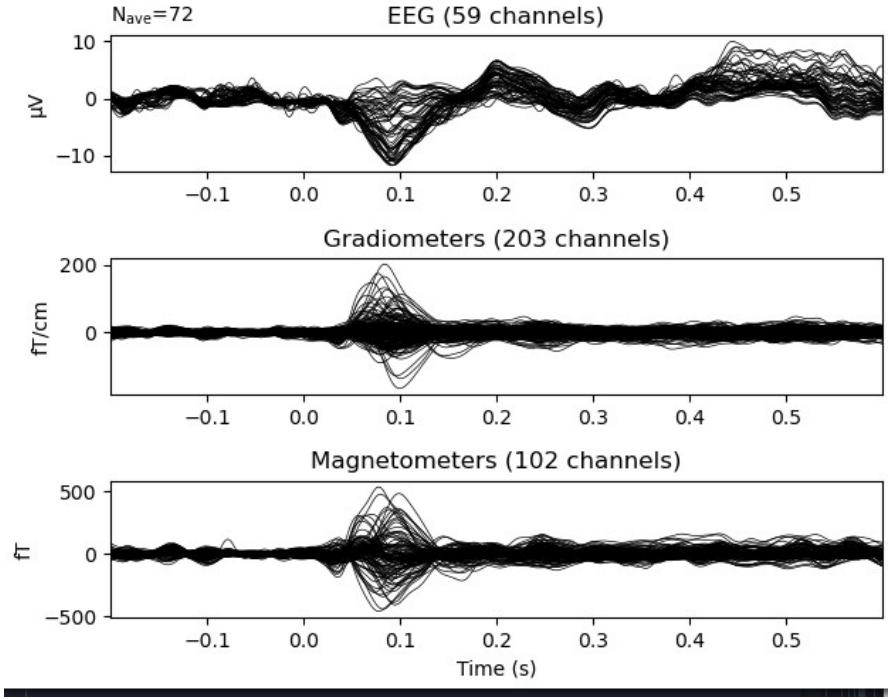
```
In [142]: print(logreg.y_predict == y_test)
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True]
```

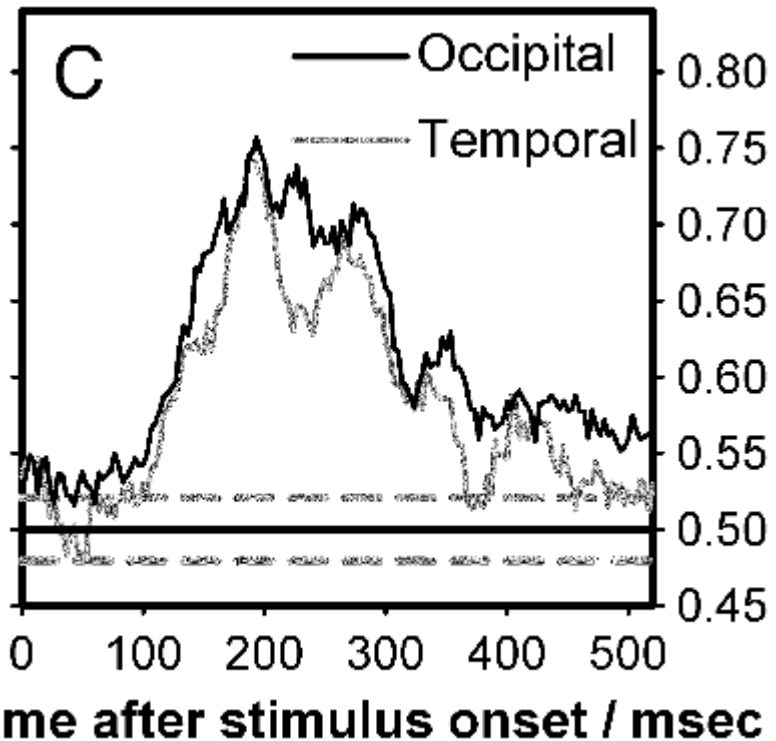


Quantiser

```
def predict(self, X):

    X = np.insert(X, 0, 1, axis=1) # adding intercept term
    yhat = np.dot(X, self.w_)
    self.y_predict = (yhat >= 0.50).astype(int)
```

Richness of neuroimaging data decoding





An MEG dataset will have ~300,000 data points per second

Reduces data markedly

Sandberg et al., 2013

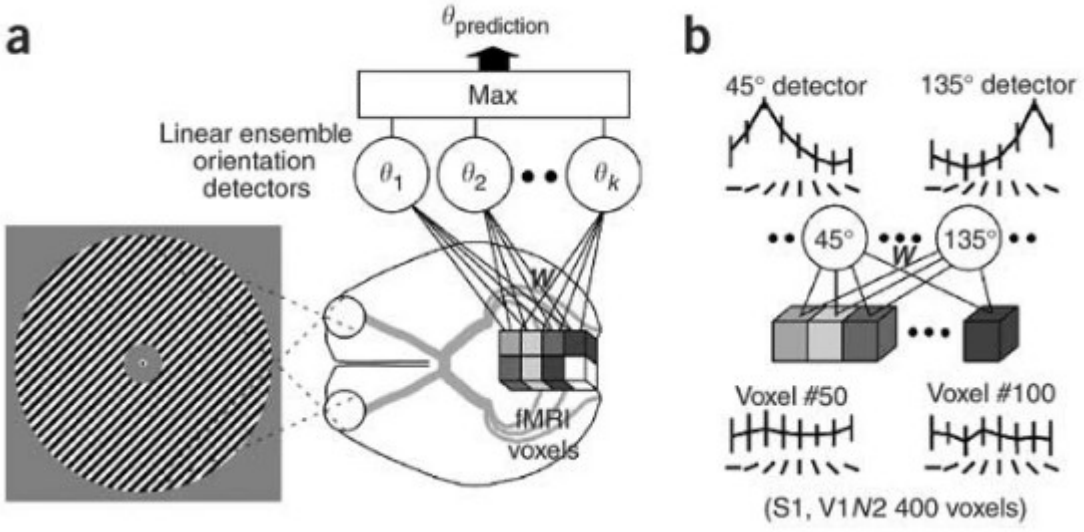
Differences

- ENCODING
  - Describes category
  - Rich data necessitates strategies for controlling the Family-wise error rate
  - Allows for inferences about specific features, e.g. *channel, voxel, frequency, time*
- DECODING
  - Predicts category
  - Rich data can be reduced to a few data points, reducing the impact of the Family-wise error rate
  - Does not (easily) allow for inferences about specific features (black box)

Interim summary

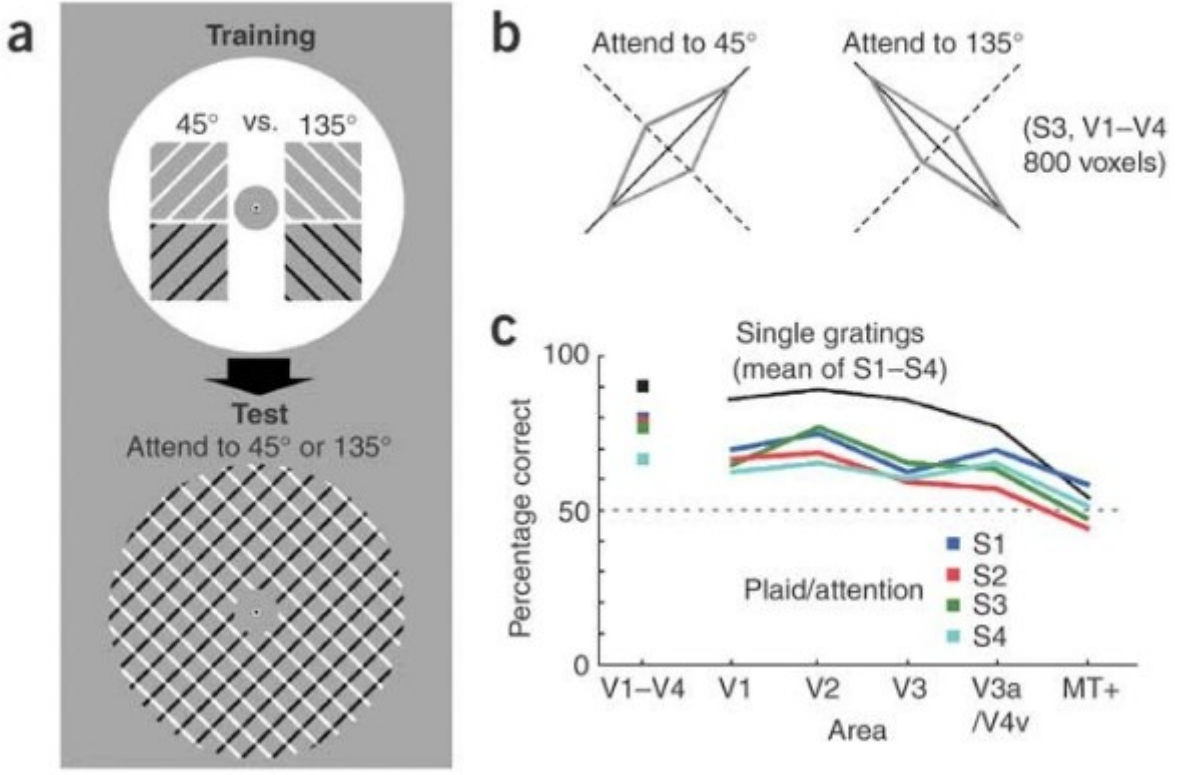
- Logistic regression can be used to reduce the richness of high-dimensional neuroimaging data
- The tools behind are basically those you (could) have learnt during Methods 3
- The features/the independent variables are the sensors
- The outcome/the dependent variable is the condition you are aiming to decode/classify

Early example – V1 encodes orientation



Kamitani and Tong, 2005

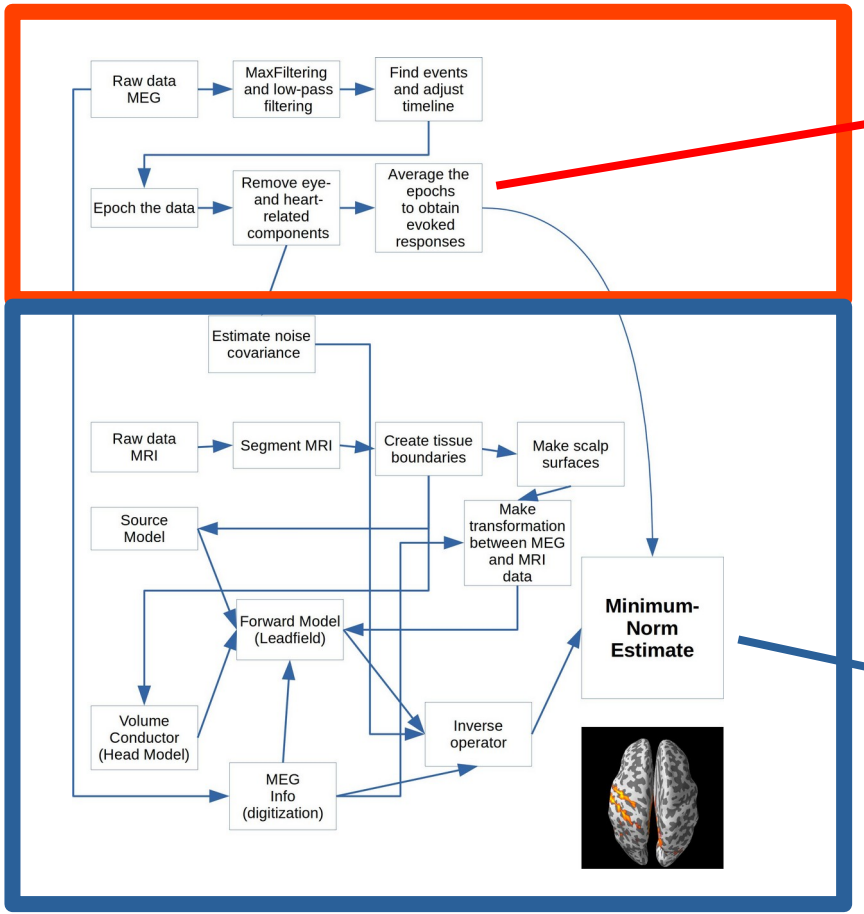
Early example – Subjective attention can be decoded



Kamitani and Tong, 2005

Sensor analysis

Source analysis



Decoding: higher accuracy; lower specificity

Andersen, 2018

Decoding: lower accuracy; higher specificity

Accuracy differences: our source space cannot explain all the variance in the sensor space; therefore accuracy is likely to be higher in the sensor space, since there is more variance available for the logistic regression to use

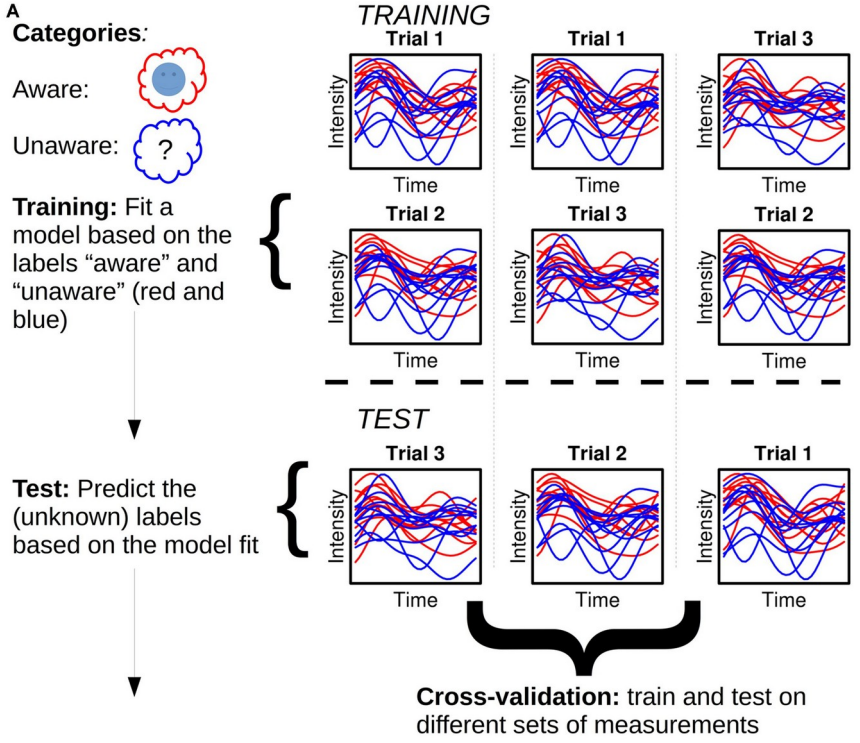
Specificity differences: in the sensor space, we cannot say with high confidence where the signal is coming from, which we can however in the source space

What are some things you could aim at decoding?

- Target:
  - Subjective experience (PAS 1-4)
    - use Subjective Response column from behavioural data and exchange target events (1 and 3) for new values
  - Correct/incorrect
    - combine *target_type* and *objective_response* in behavioural data and exchange target events (1 and 3) for new values
  - Left/right tilt
    - use the event codes 1 and 3 that are already there
  - Any combination of the three above

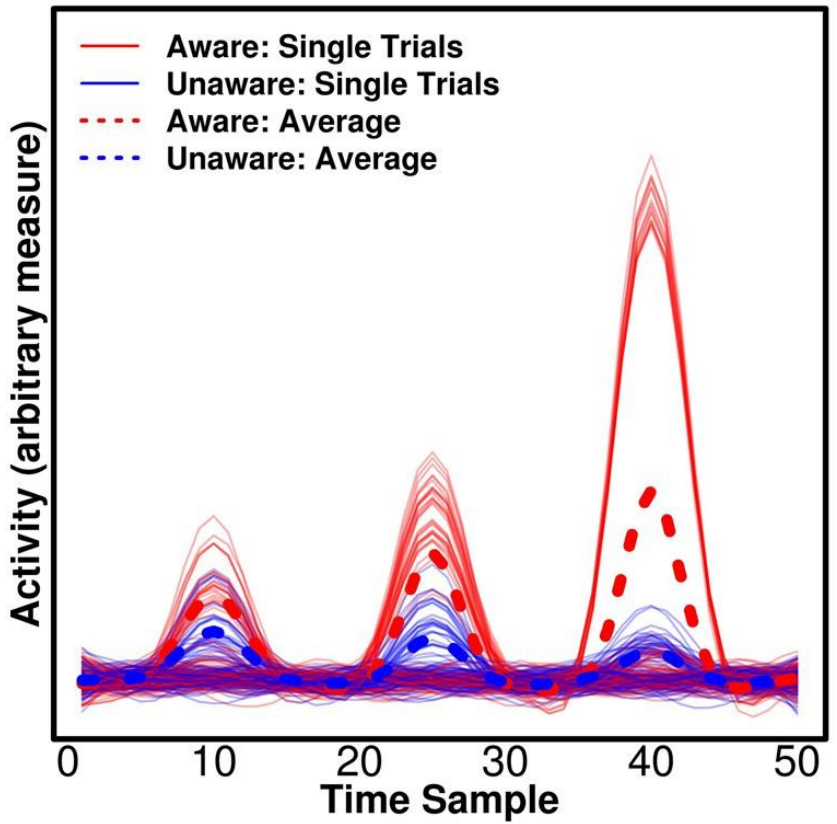
Strategy for Neural correlate of consciousnes

Sandberg et al., 2014



Valuing consistency

Sandberg et al., 2014



Exempli gratia (e.g.)

```
#%% FIND EVENTS

events = mne.find_events(raw)
print(events)
target_indices = events[:, 2] < 4
events[target_indices, 2] = behaviour['subjective_response'] + 20 ## this changes all target triggers to the subjective experience + 20
## thus ...
event_id = dict(NE=21, WG=22, ACE=23, CE=24) ## make sure it does not overlap with triggers you might already have

# Check the set_experiment parameters methods in the class Experiment
## https://github.com/ualsbombe/2025_advanced_cognitive_neuroscience/blob/main/experiment/subjective_experience_v2.py
```

What are some subsets of sensors that may be interesting?

- Occipital sensors?
- Frontal sensors?

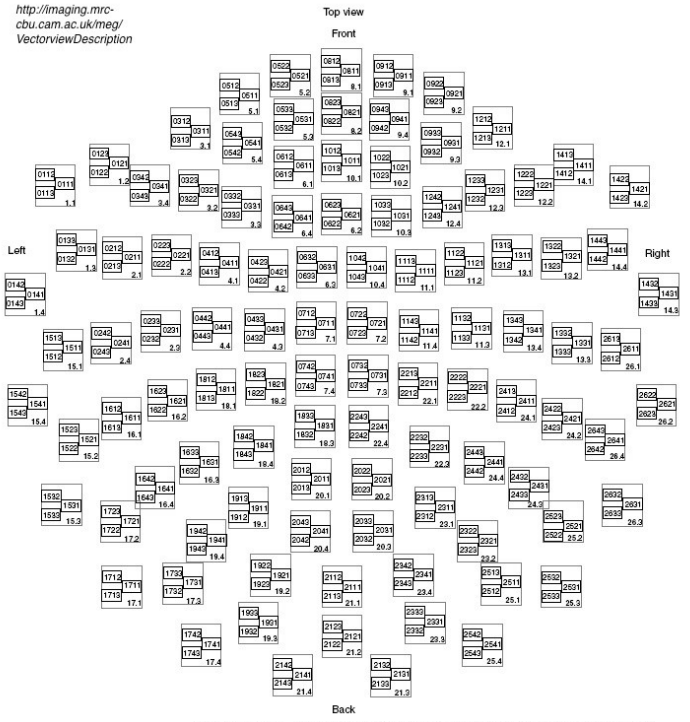
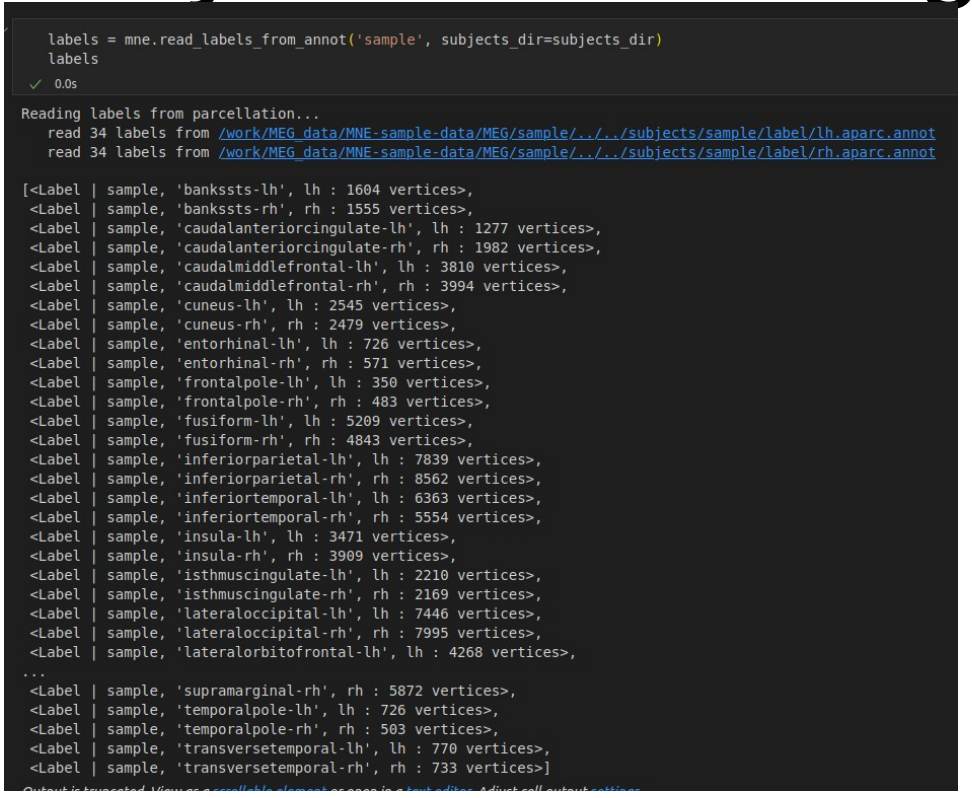


Figure 1.8.a. Layout of the sensor elements. The helmet shaped sensor array is flattened into a plane. For naming convention and gradient direction, see Fig. 1.8.b.

https://imaging.mrc-cbu.cam.ac.uk/meg/VectorviewDescription

What are some source labels that may be interesting



```
labels = mne.read_labels_from_annot('sample', subjects_dir=subjects_dir)
labels
```

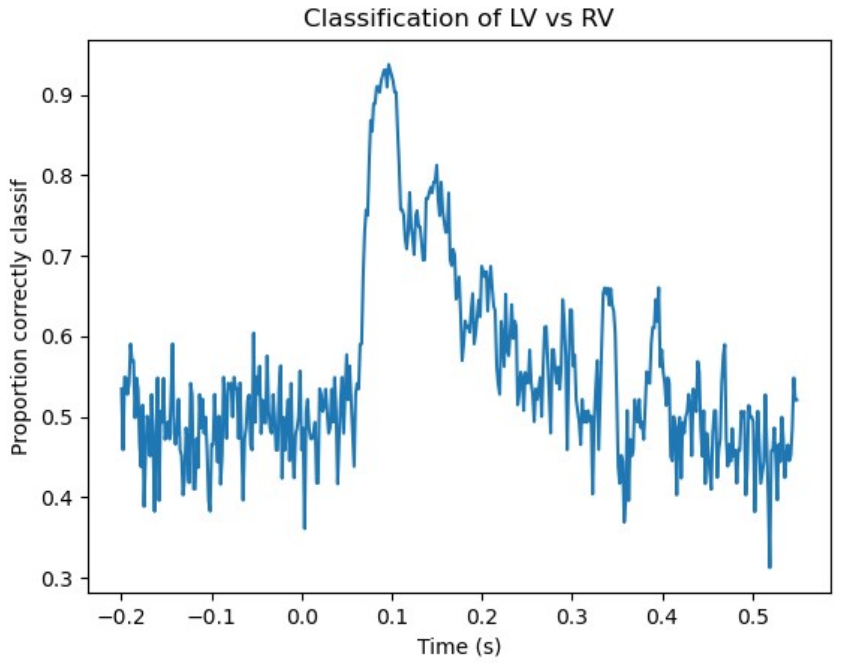
✓ 0.0s

```
Reading labels from parcellation...
   read 34 labels from /work/MEG_data/MNE-sample-data/MEG/sample/../../subjects/sample/label/lh.aparc.annot
   read 34 labels from /work/MEG_data/MNE-sample-data/MEG/sample/../../subjects/sample/label/rh.aparc.annot

[<Label | sample, 'bankssts-lh', lh : 1604 vertices>,
 <Label | sample, 'bankssts-rh', rh : 1555 vertices>,
 <Label | sample, 'caudalanteriorcingulate-lh', lh : 1277 vertices>,
 <Label | sample, 'caudalanteriorcingulate-rh', rh : 1982 vertices>,
 <Label | sample, 'caudalmiddlefrontal-lh', lh : 3810 vertices>,
 <Label | sample, 'caudalmiddlefrontal-rh', rh : 3994 vertices>,
 <Label | sample, 'cuneus-lh', lh : 2545 vertices>,
 <Label | sample, 'cuneus-rh', rh : 2479 vertices>,
 <Label | sample, 'entorhinal-lh', lh : 726 vertices>,
 <Label | sample, 'entorhinal-rh', rh : 571 vertices>,
 <Label | sample, 'frontalpole-lh', lh : 350 vertices>,
 <Label | sample, 'frontalpole-rh', rh : 483 vertices>,
 <Label | sample, 'fusiform-lh', lh : 5209 vertices>,
 <Label | sample, 'fusiform-rh', rh : 4843 vertices>,
 <Label | sample, 'inferiorparietal-lh', lh : 7839 vertices>,
 <Label | sample, 'inferiorparietal-rh', rh : 8562 vertices>,
 <Label | sample, 'inferiortemporal-lh', lh : 6363 vertices>,
 <Label | sample, 'inferiortemporal-rh', rh : 5554 vertices>,
 <Label | sample, 'insula-lh', lh : 3471 vertices>,
 <Label | sample, 'insula-rh', rh : 3909 vertices>,
 <Label | sample, 'isthmuscingulate-lh', lh : 2210 vertices>,
 <Label | sample, 'isthmuscingulate-rh', rh : 2169 vertices>,
 <Label | sample, 'lateraloccipital-lh', lh : 7446 vertices>,
 <Label | sample, 'lateraloccipital-rh', rh : 7995 vertices>,
 <Label | sample, 'lateralorbitofrontal-lh', lh : 4268 vertices>,
...
 <Label | sample, 'supramarginal-rh', rh : 5872 vertices>,
 <Label | sample, 'temporalpole-lh', lh : 726 vertices>,
 <Label | sample, 'temporalpole-rh', rh : 503 vertices>,
 <Label | sample, 'transversetemporal-lh', lh : 770 vertices>,
 <Label | sample, 'transversetemporal-rh', rh : 733 vertices>]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Note that we (generally) want to fit a model per time point

```python
#%% do logistic regression per time sample

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.preprocessing import StandardScaler

scores_list_samples = [None] * n_samples

sc = StandardScaler()

for sample_index in range(n_samples):
    print(sample_index)
    this_X = X[:, :, sample_index]
    this_X_std = sc.fit_transform(this_X) ## standardise the data
    logr = LogisticRegression(C=1e-3)
    scores_list_samples[sample_index] = np.mean(cross_val_score(logr,
                                                this_X_std,
                                                y,
                                                cv=StratifiedKFold()))
```

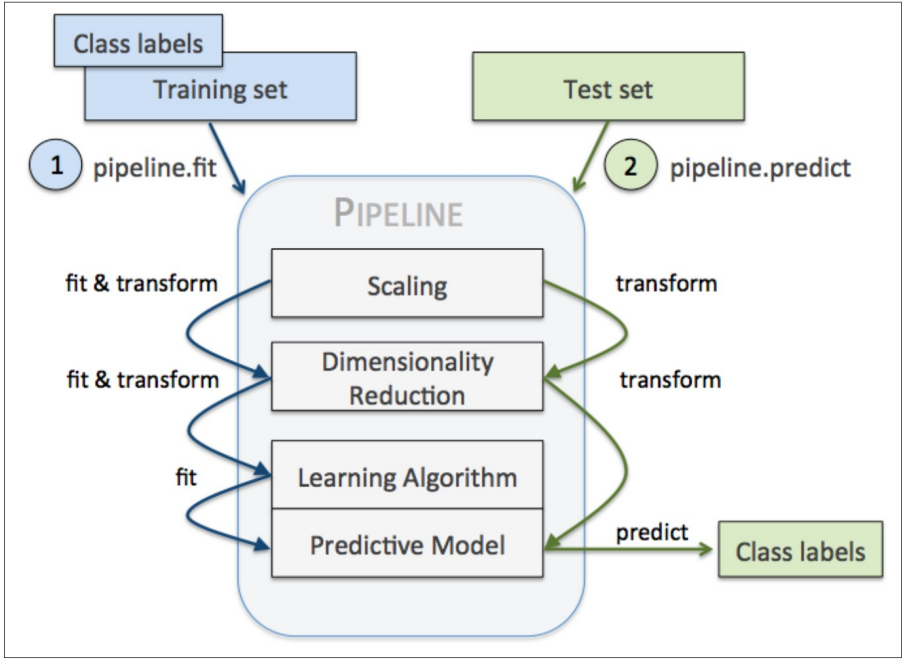



```
this_X.shape
(144, 87)
```

```
y.shape
(144,)
```

How many features/independent variables and how many repetitions?

Pipelines (scikit-learn)




```
#%% PIPELINE WAY

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model  import LogisticRegression
from sklearn.pipeline      import Pipeline

pipe_lr = Pipeline(
    [
         ('scl', StandardScaler()),
         ('pca', PCA(n_components=2)),
         ('clf', LogisticRegression(random_state=1,
                                    penalty='l2',
                                    tol=1e-4,
                                    C=1.0))
    ]
                  )

pipe_lr.fit(X_train, y_train)
print('Training accuracy: %.3f' %  pipe_lr.score(X_train, y_train))
print('Test accuracy: %.3f'     %  pipe_lr.score(X_test, y_test))
```

What does the pipeline do?

SCALING

```
#%% scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
## why transform below and not fit_transform??
X_test_std  = sc.transform(X_test)
```

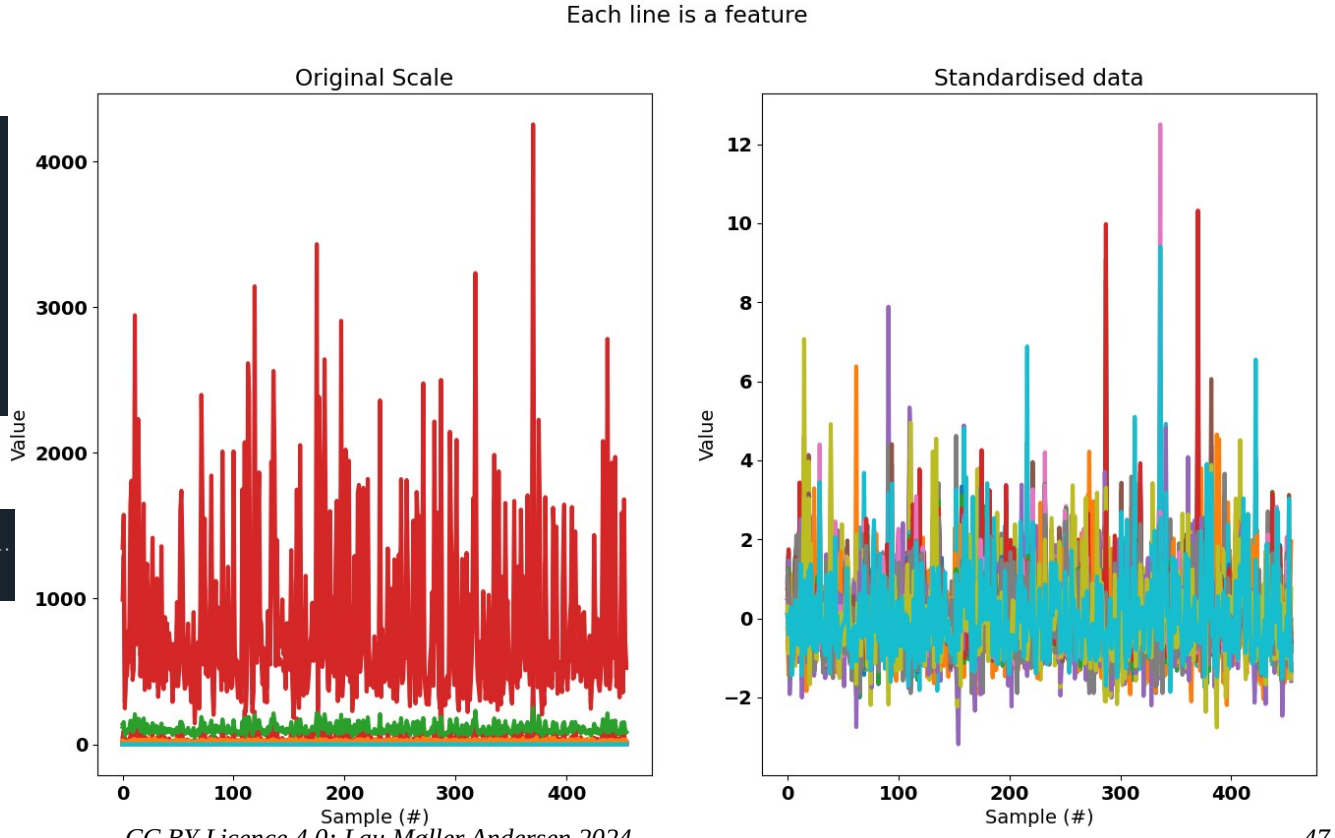
What does the pipeline do?

SCALING

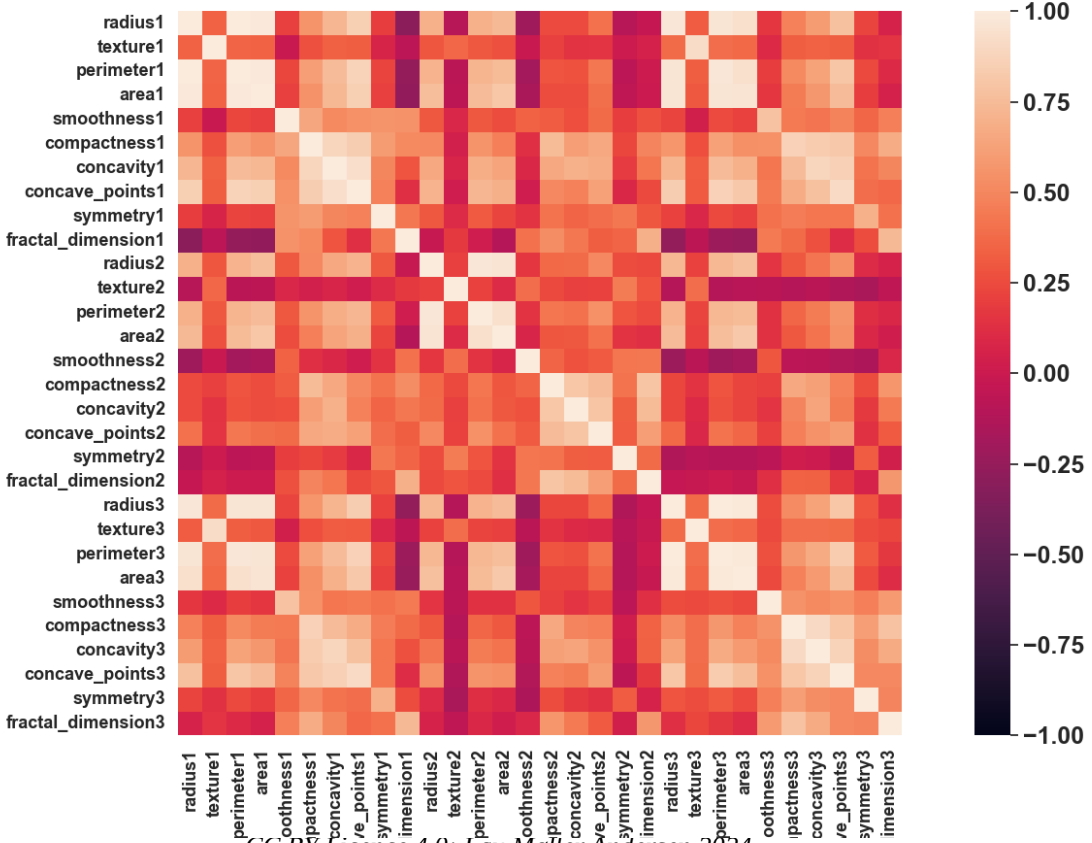
$\mu \approx 0; \sigma \approx 1$

```
In [91]: print(np.mean(X_train_std, axis=0))
[-9.46251624e-16  9.45641611e-16  4.83276423e-15  1.52259158e-16
 -3.93909569e-15 -1.31762733e-17  1.08826257e-16  1.02652929e-15
 -4.70685761e-15 -6.16966795e-15  3.64299555e-16 -6.03180509e-16
  7.99116573e-16 -4.17980668e-16 -1.72999588e-16  1.53235178e-16
  7.90576396e-17 -1.05487963e-15 -4.45065230e-16  8.61581868e-16
  3.64787565e-16  2.57193534e-15 -1.02799332e-15  8.93058521e-17
  7.92918844e-15  9.87244474e-16 -6.05132550e-16  5.52183452e-16
  7.50071556e-16  1.07996640e-15]

In [92]: print(np.mean(X_train_std, axis=0).shape)
(30,)
```

```
In [93]: print(np.std(X_train_std, axis=0))
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
```



Normalised covariance matrix

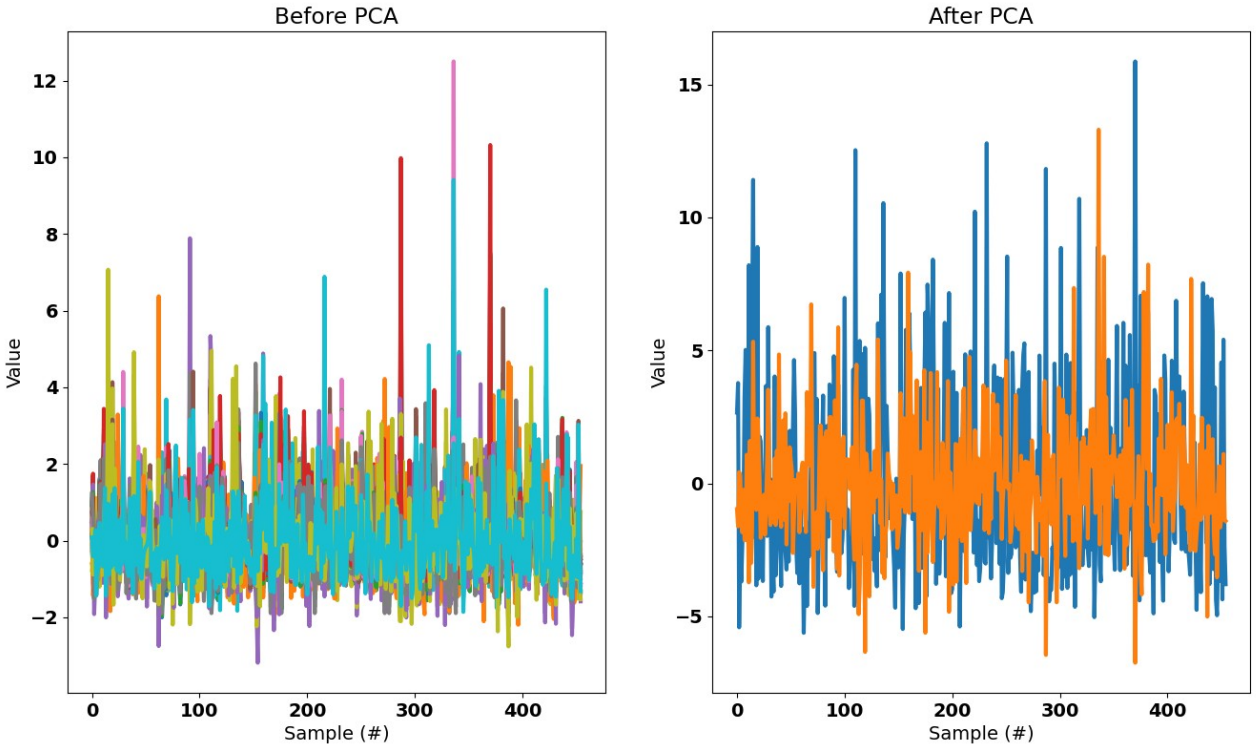


What does the pipeline do?

PCA

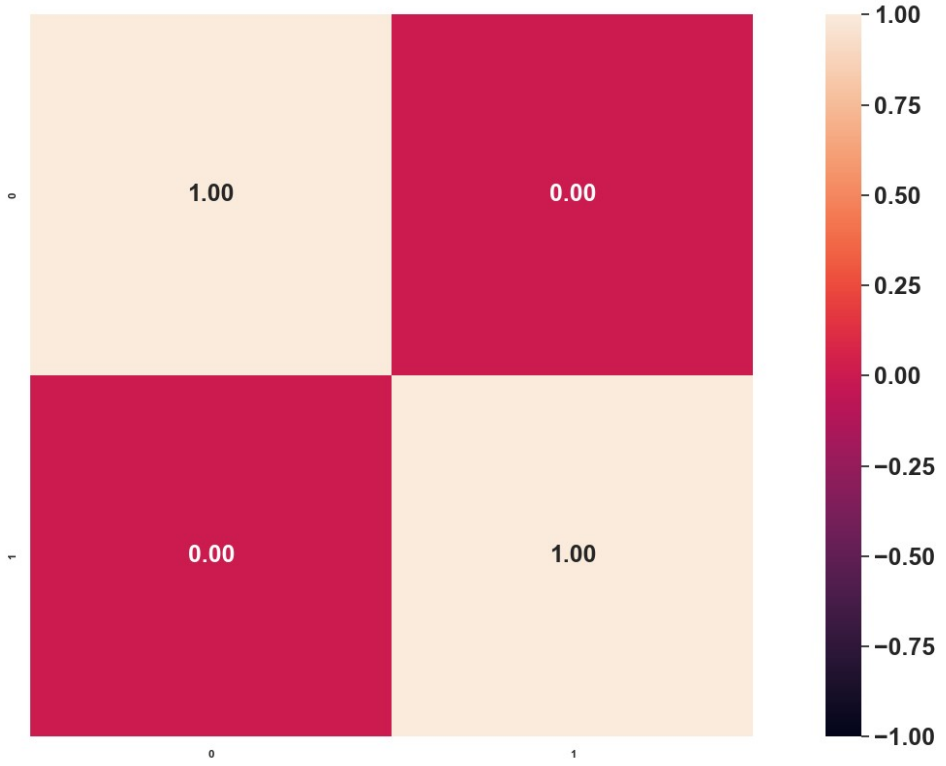
```
In [99]: print(X_train_std_pca.shape)
(455, 2)

In [100]: print(np.std(X_train_std_pca, axis=0))
[3.68403185 2.31151566]

In [131]: print(np.mean(X_train_std_pca, axis=0))
[6.24652955e-17 3.90408097e-17]
```

Each line is a feature



Normalised covariance matrix

PCA: 2 FEATURES



Normalised covariance matrix

PCA: 30 FEATURES

```
In [311]: print(np.std(X_train_std_pca, axis=0))
[3.68403185 2.31151566 1.70136353 1.39791498 1.27682331 1.11244795
 0.8320431  0.69790209 0.63149539 0.59103773 0.53860885 0.50190151
 0.4855613  0.40535005 0.30266507 0.28424771 0.24967564 0.2331132
 0.21356572 0.17574297 0.17042989 0.16936146 0.15840141 0.13523772
 0.12676504 0.0857295  0.07871836 0.03945616 0.0253066
0.01109384]
```

What does the pipeline do?
LOGISTIC REGRESSION

```
#%%# logistic regression
from sklearn.linear_model  import LogisticRegression
logreg = LogisticRegression(random_state=1, penalty='l2', C=1.0, tol=1e-4)
logreg.fit(X_train_std_pca, y_train)

print('Training accuracy: %.3f' % logreg.score(X_train_std_pca, y_train))
print('Test accuracy: %.3f' % logreg.score(X_test_std_pca, y_test))
```

Remember you want to apply the pipeline for each time sample

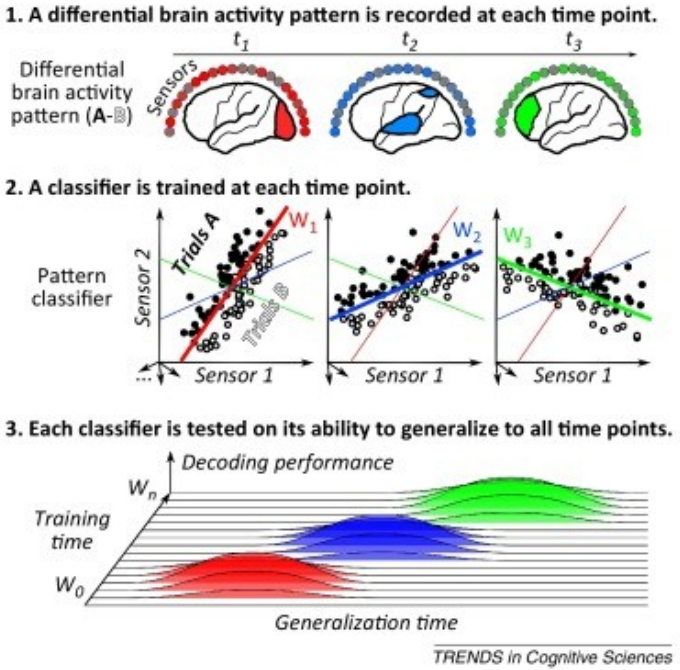
```
Training accuracy: 0.952
Test accuracy: 0.947
```

DISCUSSION POINT

Which are the *hyperparameters* here?
And why are they called *hyperparameters?*

```
#%% PIPELINE WAY

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model  import LogisticRegression
from sklearn.pipeline      import Pipeline

pipe_lr = Pipeline(
    [
         ('scl', StandardScaler()),
         ('pca', PCA(n_components=2)),
         ('clf', LogisticRegression(random_state=1,
                                    penalty='l2',
                                    tol=1e-4,
                                    C=1.0))
    ]
                )

pipe_lr.fit(X_train, y_train)
print('Training accuracy: %.3f' %  pipe_lr.score(X_train, y_train))
print('Test accuracy: %.3f'     %  pipe_lr.score(X_test, y_test))
```

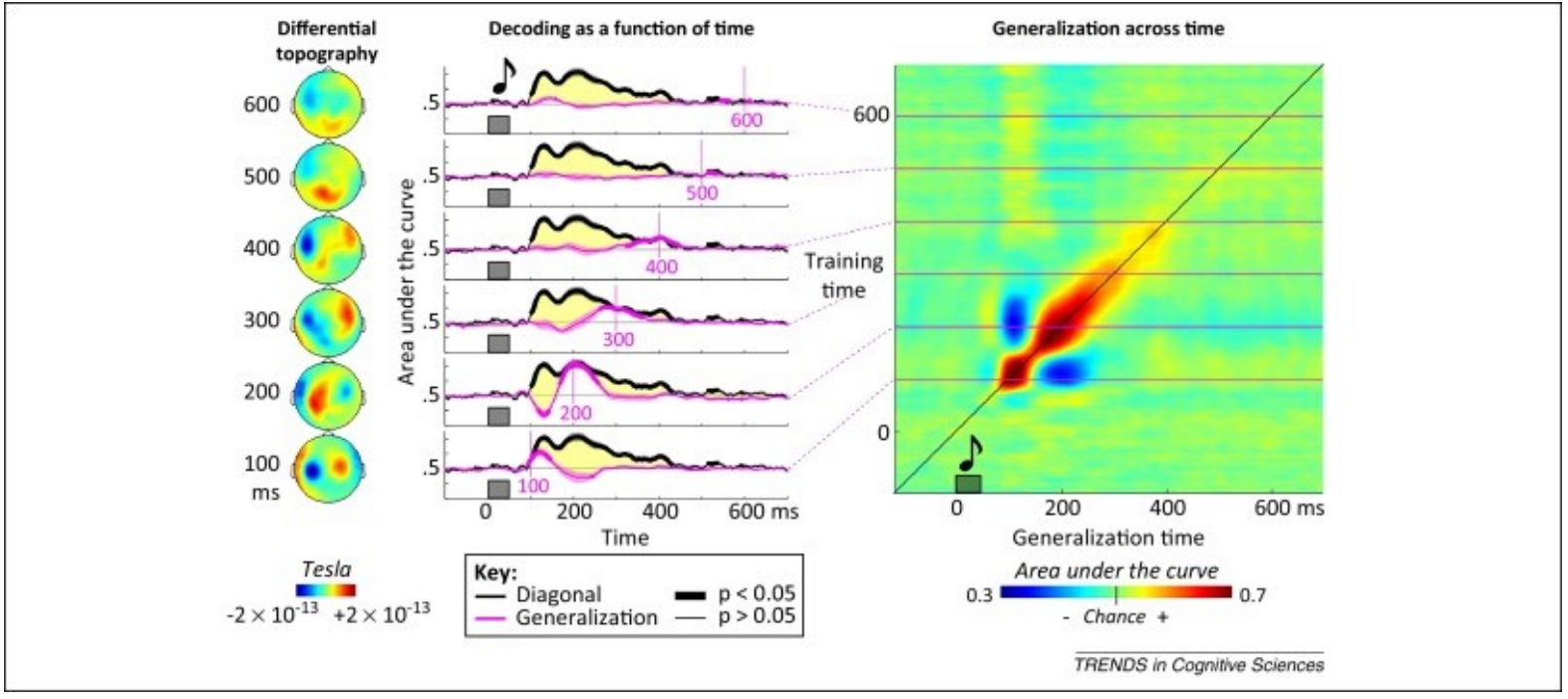
More advanced stuff



(King and Dehaene 2014)

Check out:
https://mne.tools/stable/auto_examples/decoding/decoding_time_generalization_conditions.html

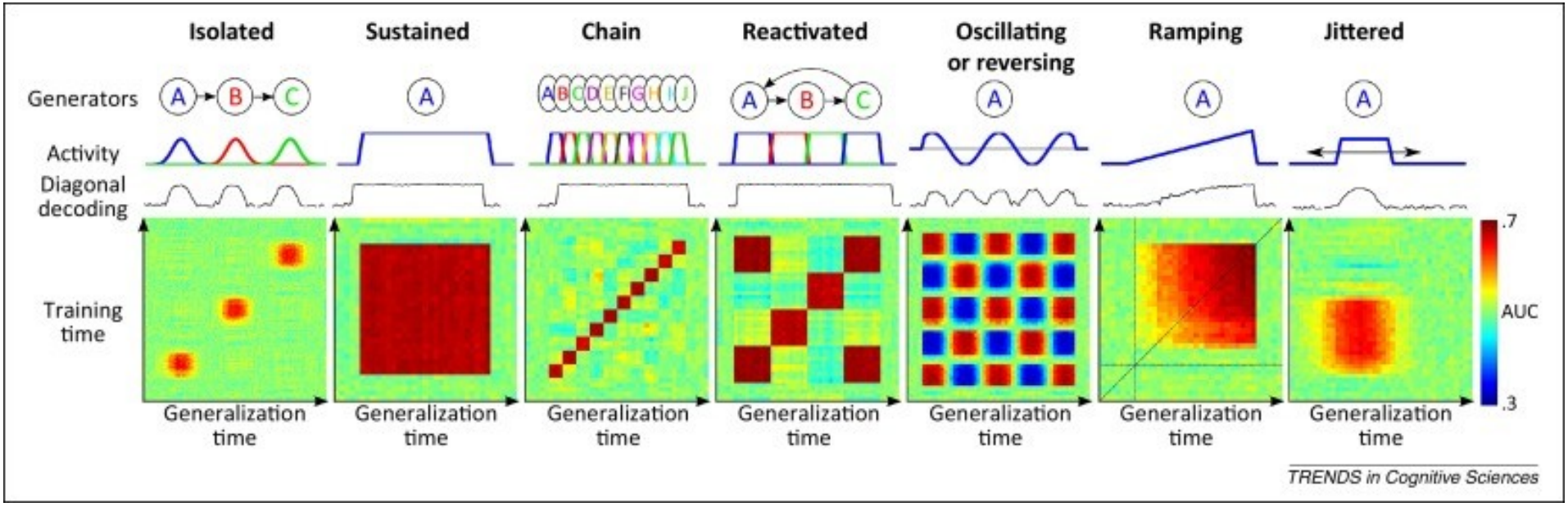
More advanced stuff



(King and Dehaene 2014)

Check out:
https://mne.tools/stable/auto_examples/decoding/decoding_time_generalization_conditions.html

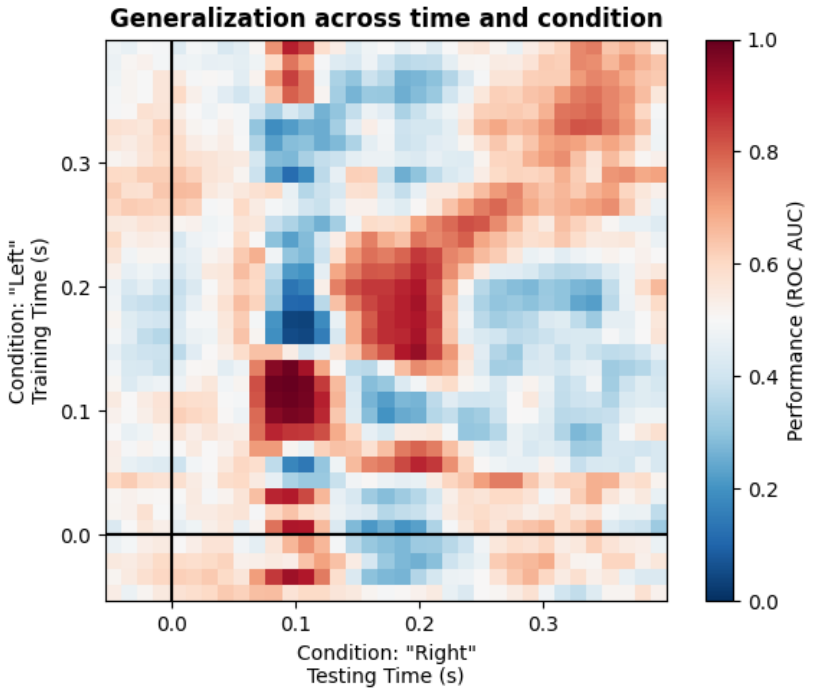
More advanced stuff



(King and Dehaene 2014)

Check out:
https://mne.tools/stable/auto_examples/decoding/decoding_time_generalization_conditions.html

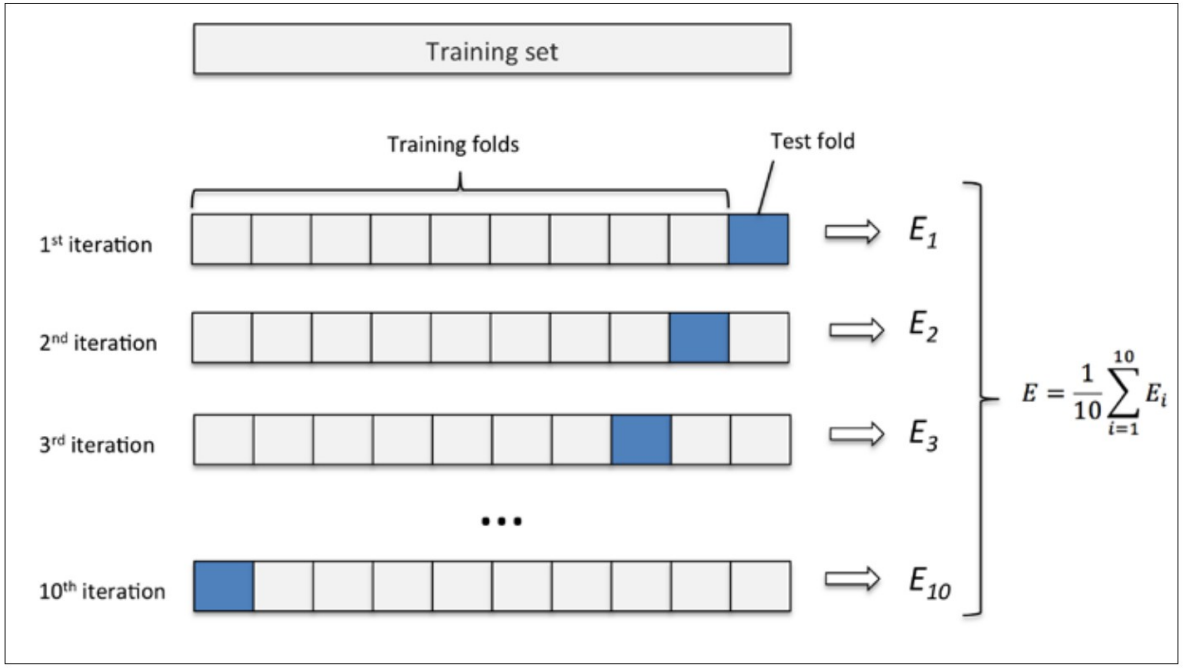
Implemented in MNE-Python



https://mne.tools/stable/auto_examples/decoding/decoding_time_generalization_conditions.html

Pimping your pipeline even further

K-fold cross validation



(p. 176: Raschka, 2015)

Stratified K-fold validation

```python
#%% STRATIFIED K-FOLD - adapted from p. 177

import numpy as np
from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=10)
scores = list() ## create an empty list

## just checking what kfold.split does
for index, (train_indices, test_indices) in enumerate(kfold.split(X, y)):
    print(f"Fold {index}:")
    print(f"Train indices: indices={train_indices[:10]}")
    print(f"Test indices: indices={test_indices[:10]}")

## actually applying it
for index, (train_indices, test_indices) in enumerate(kfold.split(X, y)):
    pipe_lr.fit(X[train_indices], y[train_indices])
    score = pipe_lr.score(X[test_indices], y[test_indices])
    scores.append(score)
    print('Fold: %s, Class dist.: %s, Acc: %.3f' % (index,
          np.bincount(y[train_indices]), score))

print('Cross-validation accuracy: %.3f +/- %.3f' % (
    np.mean(scores), np.std(scores)))
```

Stratified – making sure that there is an equal count of each category for each split – check the output of *np.bincount*

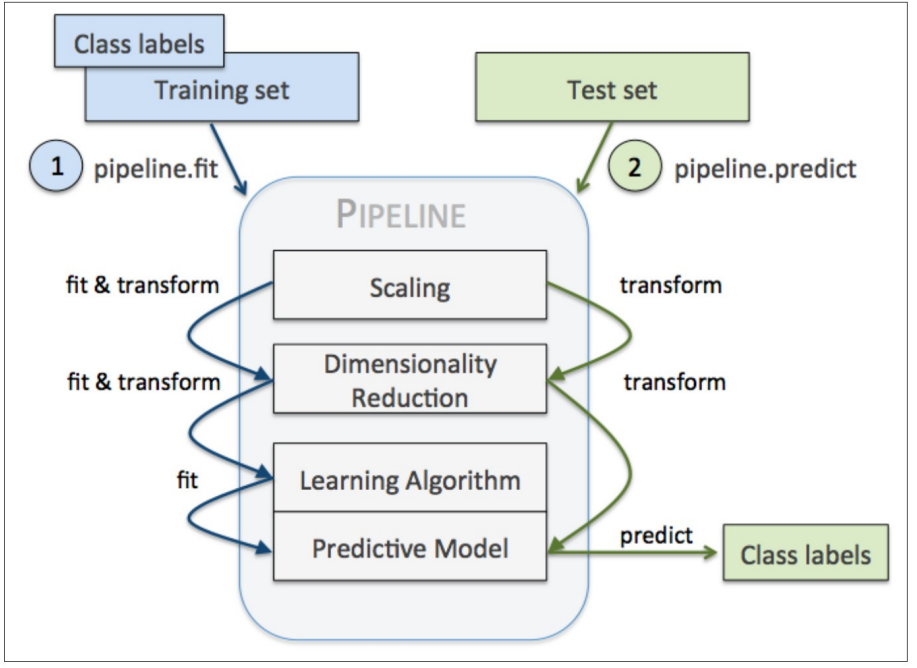
A slight improvement over the standard k-fold cross-validation approach is stratified k-fold cross-validation, which can yield better bias and variance estimates, especially in cases of unequal class proportions, as it has been shown in a study by R. Kohavi et al. (R. Kohavi et al. *A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection*. In Ijcai, volume 14, pages 1137–1145, 1995). In stratified cross-validation, the class proportions are preserved in each fold to ensure that each fold is representative of the class proportions in the training dataset, which we will illustrate by using the `StratifiedKFold` iterator in scikit-learn:

(p. 176: Raschka, 2015)

```
#%% IMPLEMENTING IT IN PIPELINE

from sklearn.model_selection import cross_val_score
scores = cross_val_score(estimator=pipe_lr, X=X, y=y, cv=10, n_jobs=1)

print('Cross-validation accuracy scores: %s' % scores)
print('Cross-validation accuracy: %.3f +/- %.3f' % (
    np.mean(scores), np.std(scores)))
```

for cv_index in range(cv):

What does *n_jobs* do?



(p. 172: Raschka, 2015)

Grid Search

```
#%% GRID SEARCH CV - setup

from sklearn.model_selection import GridSearchCV
param_range_C   = [1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4]
param_range_PCA = np.arange(1, 15)
## Dictionary entry: name, e.g. 'clf' what it was called in the pipeline,
## two underscores and then the name of the parameter, e.g. C or n_components
param_grid = [{'clf__C':            param_range_C, # you need two underscores
               'pca__n_components': param_range_PCA}]

grid_search = GridSearchCV(estimator=pipe_lr, param_grid=param_grid,
                           cv=10, n_jobs=-1) ## will use StratifiedKFold
```

```
#%% GRID SEARCH CV - fit

grid_search.fit(X, y)
print(grid_search.best_params_)
```

```
{'clf__C': 1.0, 'pca__n_components': 9}
```

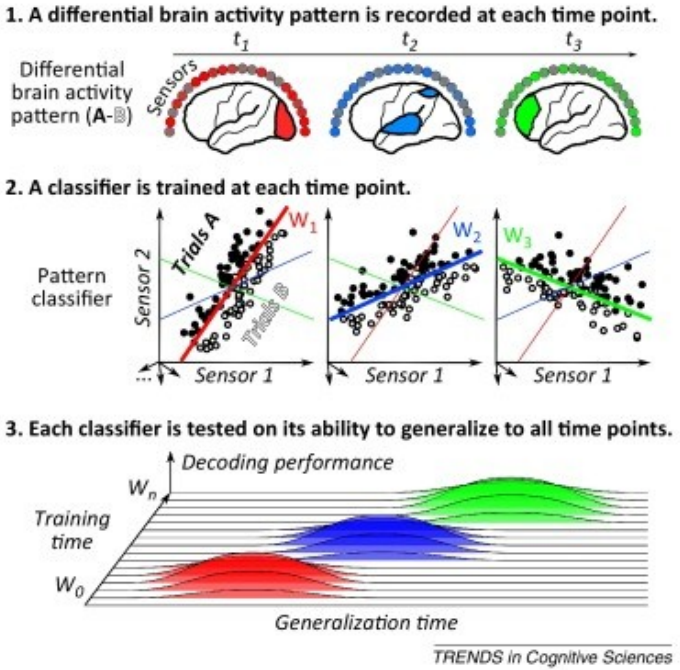
Randomised grid search

Although grid search is a powerful approach for finding the optimal set of parameters, the evaluation of all possible parameter combinations is also computationally very expensive. An alternative approach to sampling different parameter combinations using scikit-learn is randomized search. Using the `RandomizedSearchCV` class in scikit-learn, we can draw random parameter combinations from sampling distributions with a specified budget. More details and examples for its usage can be found at `http://scikit-learn.org/stable/modules/grid_search.html#randomized-parameter-optimization`.

```
# specify parameters and distributions to sample from
param_dist = {
    "average": [True, False],
    "l1_ratio": stats.uniform(0, 1),
    "alpha": stats.loguniform(1e-2, 1e0),
```

https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html#sphx-glr-auto-examples-model-selection-plot-randomized-search-py

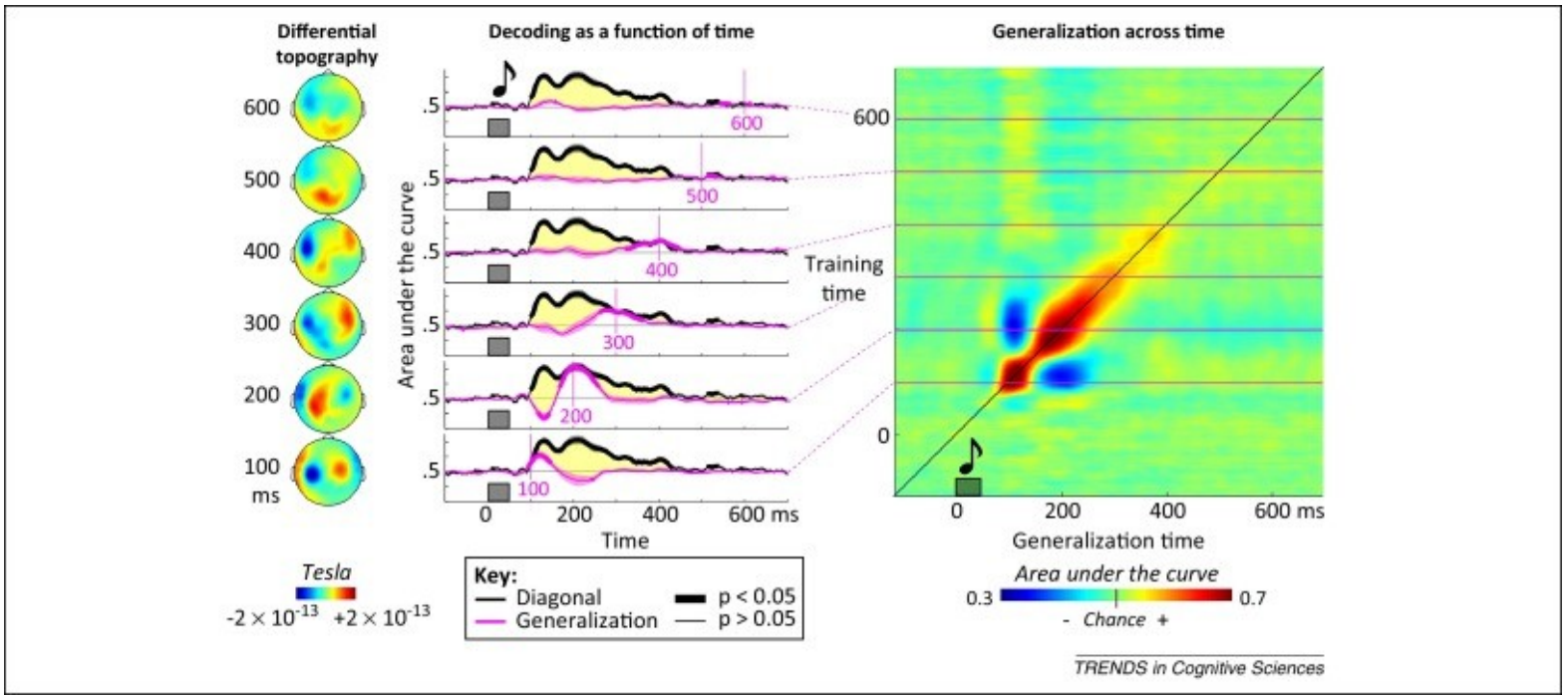
More advanced stuff



(King and Dehaene 2014)

Check out:
https://mne.tools/stable/auto_examples/decoding/decoding_time_generalization_conditions.html

More advanced stuff

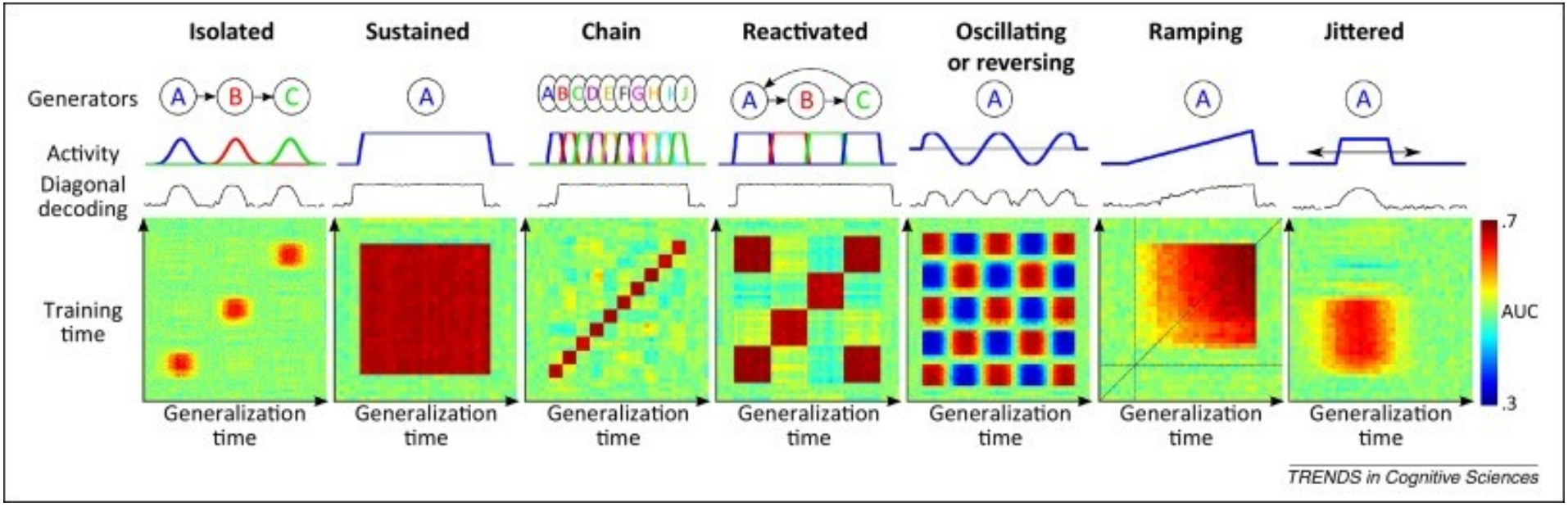


(King and Dehaene 2014)

Check out:

https://mne.tools/stable/auto_examples/decoding/decoding_time_generalization_conditions.html

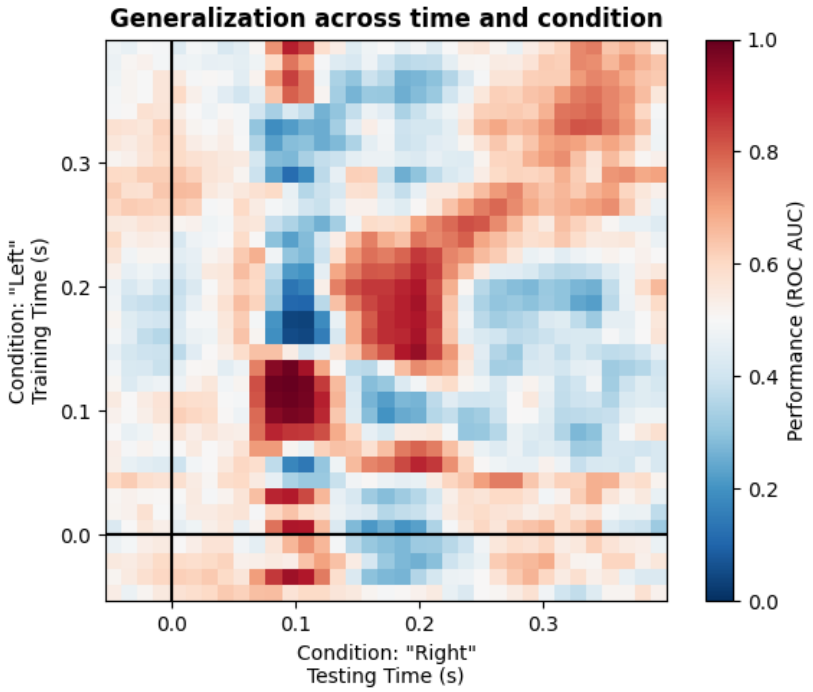
More advanced stuff



(King and Dehaene 2014)

Check out:
https://mne.tools/stable/auto_examples/decoding/decoding_time_generalization_conditions.html

Implemented in MNE-Python



https://mne.tools/stable/auto_examples/decoding/decoding_time_generalization_conditions.html

Summary

- With *pipelines*, we can specify our pipelines in a controlled high-level fashion, while testing out all hyperparameters
- There are many ways you can do your analysis
  - I know this is a lot of freedom, but I have only shown logistic regression; you may even use other classifiers

Learning goals

- Did you (re)learn:
  - the basics of logistic regression
  - what targets could be aimed at in our experiment
  - what the pros and cons of multivariate statistics are for MEG data
    - what are the differences between sensor space decoding and source space decoding
- How to set up a pipeline
  - Standardisation
  - Feature selection (e.g. PCA)
  - Finding optimal hyperparameters (e.g. Grid Search)

The course plan

Week 36:
Lesson 0: What is it all about?
Class 0: Setting up UCloud and installing MNE-Python

Week 37:
No Teaching

Week 38:
Lesson 1: Workshop paradigm: Measuring visual subjective experience + MR Recordings
Class 1: Running an MEG analysis of visual responses

Week 39:
MEG workshop: Measuring and predicting visual subjective experience

Week 40:
Lesson 2: Basic physiology and Evoked responses
Class 2: Evoked responses to different levels of subjective experience

Week 41:
Lesson 3: Multivariate statistics
Class 3: Predicting subjective experience in sensor space
Deadline for feedback: Video Explainer

Week 42:
Autumn Break

Week 43:
Lesson 4: Forward modelling and dipole estimation
Class 4: Creating a forward model

Week 44:
Lesson 5: Inverse modelling: Minimum-norm estimate
Class 5: Predicting subjective experience in source space

Week 45:
Lesson 6: Inverse modelling: Beamforming
Class 6: Predicting subjective experience in source space, continued

Week 46:
Lesson 7: What about that other cortex? - the cerebellar one
Class 7: Oral presentations (part 1)
Deadline for feedback: Lab report

Week 47:
Lesson 8: Guest lecture: Laura Bock Paulsen: Respiratory analyses
Class 8: Oral presentations (part 2)

Week 48:
Lesson 9: Guest lecture: Barbara Pomiechowska: Using OPM-MEG to study brain and cognitive development in infancy
Class 9: Oral presentations (part 3)

Week 49:
Lesson 0 again: What was it all about?
Class 10: Oral presentations (part 4)

Reading questions

- In chapter 2, focus on sections 2.1 and 2.2, but do also read section 2.5
- FieldTrip Video
  - What does superposition of source activity mean?
  - Why is the forward model a simulation?
  - How is the localisation of the fish similar to MEG source localisation

Next class – Continuing with your own data