In [59]:
```
'''
This jupyter notebook includes code taken from the following nilearn project,
"8.3.10. Voxel-Based Morphometry on Oasis dataset", which predicts age from
grey matter morhpometry. Feature redux = k-best ANOVA, prediction function = SVM

I am tweaking the code to the following analysis:
Feature redux = PCA, prediction function = SVM.
I've included various plots.
'''
```

In [ ]:
```
# Let's keep our notebook clean, so it's a little more readable!
import warnings
warnings.filterwarnings('ignore')
```

In [99]:
```
from nilearn import datasets
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
from nilearn.input_data import NiftiMasker
```

In [60]:
```
from nilearn import plotting
```

In [53]:
```
oasis_dataset = datasets.fetch_oasis_vbm()
gray_matter_map_filenames = oasis_dataset.gray_matter_maps
age = oasis_dataset.ext_vars['age'].astype(float, None)
```

In [74]:
```
#Checking to see where the dataset is downloaded onto my computer
oasis_dataset
```

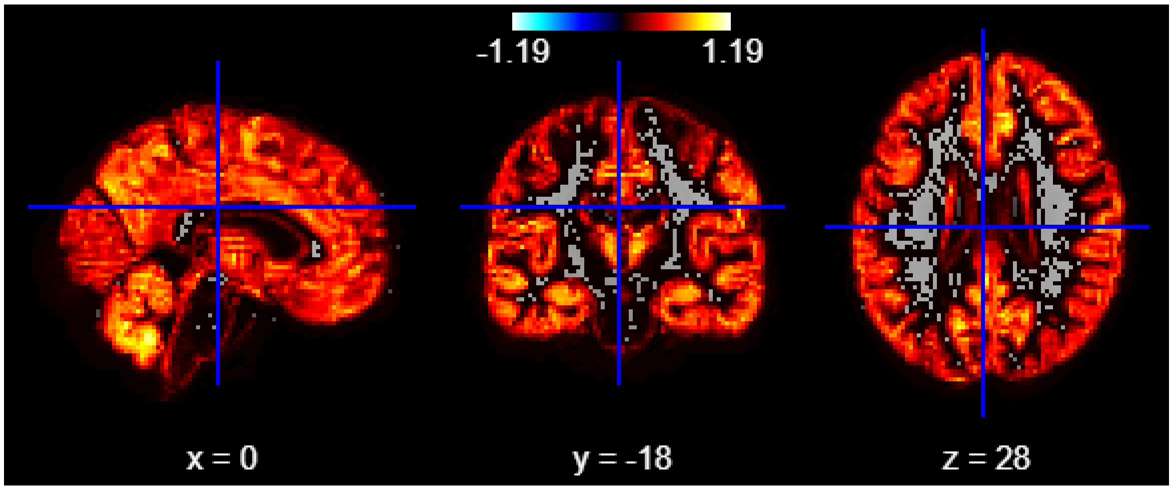
Out[74]:
```
{'gray_matter_maps': ['C:\\Users\\rwick/nilearn_data\\oasis1\\OAS1_0001_MR1
\\mwrc1OAS1_0001_MR1_mpr_anon_fslswapdim_bet.nii.gz',
  'C:\\Users\\rwick/nilearn_data\\oasis1\\OAS1_0002_MR1\\mwrc1OAS1_0002_MR1_
mpr_anon_fslswapdim_bet.nii.gz',
  'C:\\Users\\rwick/nilearn_data\\oasis1\\OAS1_0003_MR1\\mwrc1OAS1_0003_MR1_
mpr_anon_fslswapdim_bet.nii.gz',
  'C:\\Users\\rwick/nilearn_data\\oasis1\\OAS1_0004_MR1\\mwrc1OAS1_0004_MR1_
mpr_anon_fslswapdim_bet.nii.gz',
  'C:\\Users\\rwick/nilearn_data\\oasis1\\OAS1_0005_MR1\\mwrc1OAS1_0005_MR1_
mpr_anon_fslswapdim_bet.nii.gz',
  'C:\\Users\\rwick/nilearn_data\\oasis1\\OAS1_0006_MR1\\mwrc1OAS1_0006_MR1_
mpr_anon_fslswapdim_bet.nii.gz',
  'C:\\Users\\rwick/nilearn_data\\oasis1\\OAS1_0007_MR1\\mwrc1OAS1_0007_MR1_
mpr_anon_fslswapdim_bet.nii.gz',
  'C:\\Users\\rwick/nilearn_data\\oasis1\\OAS1_0009_MR1\\mwrc1OAS1_0009_MR1_
mpr_anon_fslswapdim_bet.nii.gz',
  'C:\\Users\\rwick/nilearn_data\\oasis1\\OAS1_0010_MR1\\mwrc1OAS1_0010_MR1_
mpr_anon_fslswapdim_bet.nii.gz',
  'C:\\Users\\rwick/nilearn_data\\oasis1\\OAS1_0011_MR1\\mwrc1OAS1_0011_MR1_
```

In [55]:
```
'''
#for later if I decide to look at white matter
#Note to self: make sure the nifti masker is adapted for white matter, with corre
white_matter_map_filenames = oasis_dataset.white_matter_maps
wm_maps_masked = nifti_masker.fit_transform(white_matter_map_filenames)
'''
```

In [ ]:
```
nifti_masker = NiftiMasker(
    standardize=False,
    smoothing_fwhm=2,
    memory='nilearn_cache')  # cache options
gm_maps_masked = nifti_masker.fit_transform(gray_matter_map_filenames)

n_samples, n_features = gm_maps_masked.shape
print("%d samples, %d features" % (n_subjects, n_features))
```

In [61]:
```
subject1 = oasis_dataset.gray_matter_maps[0]
plotting.view_img(subject1)
```

Out[61]:

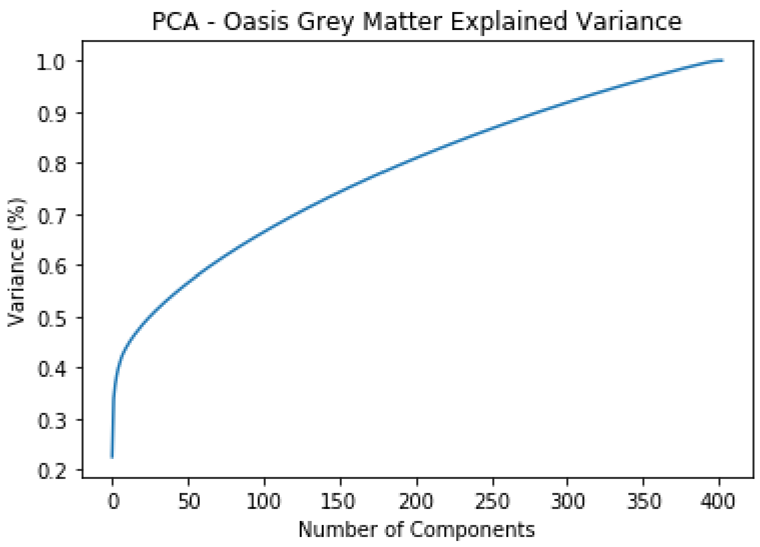


In [58]:
```
print("ANOVA + SVR")
# Define the prediction function to be used.
# Here we use a Support Vector Classification, with a linear kernel
from sklearn.svm import SVR
svr = SVR(kernel='linear')

# Dimension reduction
from sklearn.feature_selection import VarianceThreshold, SelectKBest, \
        f_regression

# Remove features with too low between-subject variance
variance_threshold = VarianceThreshold(threshold=.01)

# Here we use a classical univariate feature selection based on F-test,
# namely Anova.
feature_selection = SelectKBest(f_regression, k=2000)

# We have our predictor (SVR), our feature selection (SelectKBest), and now,
# we can plug them together in a *pipeline* that performs the two operations
# successively:
from sklearn.pipeline import Pipeline
anova_svr = Pipeline([
            ('variance_threshold', variance_threshold),
            ('anova', feature_selection),
            ('svr', svr)])

### Fit and predict
anova_svr.fit(gm_maps_masked, age)
age_pred = anova_svr.predict(gm_maps_masked)
```

```
ANOVA + SVR
```

In [131]:
```
print("PCA + SVR")
# Define the prediction function to be used.
# Here we use a Support Vector Classification, with a linear kernel
from sklearn.svm import SVR
svr = SVR(kernel='linear')

# Dimension reduction

from sklearn.decomposition import PCA
#from sklearn.preprocessing import MinMaxScaler -
#not needed because values are between 0 and 1, as they are tissue probabilities
pca = PCA() # Here I'm not setting any hyperparameters on the number of component
pca.fit(gm_maps_masked) # Fit the data.

#These components are in an array called pca.explained_variance_ratio_
#What if I want to see factor loadings? This would basically show every single vo
#and how it correlates with each loading, Which would be a huge figure.
#What if I want to see a covariance/correlation matrix between all of my componen

#Plotting the Cumulative Summation of the Explained Variance
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('PCA - Oasis Grey Matter Explained Variance')
plt.show()

#Scree plot: a line plot of the eigenvalues of factors or principal components in
#Try later

'''
We have our predictor (SVR), our feature selection (PCA), and now,
we can plug them together in a *pipeline* that performs the two operations
successively:
'''

# Remove components with too low between-subject variance
variance_threshold = VarianceThreshold(threshold=.001)

from sklearn.pipeline import Pipeline
pca_svr = Pipeline([
            ('variance_threshold', variance_threshold),
            ('PCA', pca),
            ('svr', svr)])

# Fit and predict
pca_svr.fit(gm_maps_masked, age)
age_pred = pca_svr.predict(gm_maps_masked)
```

```
PCA + SVR
```


PCA - Oasis Grey Matter Explained Variance
Variance (%)
Number of Components
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
50
100
150
200
250
300
350
400

In [130]:
```
pca.explained_variance_ratio_
```

Out[130]:
```
array([2.2584088e-01, 1.1436371e-01, 2.6626622e-02, 1.7880067e-02,
       1.3828022e-02, 1.0346623e-02, 9.5663518e-03, 7.4550672e-03,
       6.2674419e-03, 5.5455039e-03, 5.1566600e-03, 5.0084377e-03,
       4.4522383e-03, 4.3732417e-03, 4.2156246e-03, 3.9440263e-03,
       3.8395098e-03, 3.8166337e-03, 3.7471692e-03, 3.5620523e-03,
       3.4652401e-03, 3.4333770e-03, 3.2579077e-03, 3.2121697e-03,
       3.1870520e-03, 3.1186503e-03, 3.0648196e-03, 2.9737179e-03,
       2.9191973e-03, 2.9087877e-03, 2.8553784e-03, 2.7970090e-03,
       2.7778735e-03, 2.7408651e-03, 2.7185453e-03, 2.6765417e-03,
       2.6360289e-03, 2.6213841e-03, 2.5917827e-03, 2.5617639e-03,
       2.5264965e-03, 2.5132613e-03, 2.5012239e-03, 2.4725255e-03,
       2.4542816e-03, 2.4079867e-03, 2.3965258e-03, 2.3721035e-03,
       2.3530673e-03, 2.3207669e-03, 2.3078641e-03, 2.2899993e-03,
       2.2746450e-03, 2.2634489e-03, 2.2366601e-03, 2.2206153e-03,
       2.2153966e-03, 2.1884532e-03, 2.1630446e-03, 2.1581594e-03,
       2.1476038e-03, 2.1241780e-03, 2.1107635e-03, 2.1010994e-03,
       2.0949480e-03, 2.0748654e-03, 2.0677606e-03, 2.0486242e-03,
       2.0426018e-03, 2.0294755e-03, 2.0136640e-03, 1.9954494e-03,
       1.9905095e-03, 1.9788677e-03, 1.9669503e-03, 1.9592512e-03,
       1.9466685e-03, 1.9379745e-03, 1.9209754e-03, 1.9200307e-03,
       1.9077482e-03, 1.8977473e-03, 1.8904781e-03, 1.8648980e-03,
       1.8602778e-03, 1.8486676e-03, 1.8398013e-03, 1.8278993e-03,
       1.8221135e-03, 1.8098885e-03, 1.8025546e-03, 1.7984129e-03,
       1.7895069e-03, 1.7816254e-03, 1.7705270e-03, 1.7677383e-03,
       1.7616448e-03, 1.7552840e-03, 1.7531823e-03, 1.7415775e-03,
       1.7400788e-03, 1.7281702e-03, 1.7195158e-03, 1.7069331e-03,
       1.7008554e-03, 1.6914945e-03, 1.6892923e-03, 1.6848481e-03,
       1.6759066e-03, 1.6709365e-03, 1.6679898e-03, 1.6567720e-03,
       1.6502545e-03, 1.6348436e-03, 1.6295209e-03, 1.6228094e-03,
       1.6214591e-03, 1.6141557e-03, 1.6052786e-03, 1.6030121e-03,
       1.5920963e-03, 1.5904651e-03, 1.5792794e-03, 1.5729720e-03,
       1.5722975e-03, 1.5686385e-03, 1.5603816e-03, 1.5568333e-03,
       1.5509462e-03, 1.5425691e-03, 1.5379243e-03, 1.5335671e-03,
       1.5296823e-03, 1.5209365e-03, 1.5156355e-03, 1.5131271e-03,
       1.5078814e-03, 1.4948997e-03, 1.4888986e-03, 1.4886067e-03,
       1.4816939e-03, 1.4724497e-03, 1.4692037e-03, 1.4660993e-03,
       1.4647440e-03, 1.4566776e-03, 1.4503871e-03, 1.4481053e-03,
       1.4456082e-03, 1.4369712e-03, 1.4313423e-03, 1.4245587e-03,
       1.4234260e-03, 1.4220214e-03, 1.4181057e-03, 1.4123137e-03,
       1.4108052e-03, 1.4052923e-03, 1.3962564e-03, 1.3945088e-03,
       1.3922962e-03, 1.3872288e-03, 1.3846006e-03, 1.3789861e-03,
       1.3745185e-03, 1.3691484e-03, 1.3637854e-03, 1.3618452e-03,
       1.3573989e-03, 1.3554659e-03, 1.3522484e-03, 1.3455915e-03,
       1.3406917e-03, 1.3390647e-03, 1.3323993e-03, 1.3307688e-03,
       1.3280221e-03, 1.3223292e-03, 1.3179997e-03, 1.3137892e-03,
       1.3106659e-03, 1.3068999e-03, 1.3033262e-03, 1.3005041e-03,
       1.2965433e-03, 1.2919662e-03, 1.2891974e-03, 1.2825729e-03,
       1.2807273e-03, 1.2772733e-03, 1.2750929e-03, 1.2723877e-03,
       1.2677106e-03, 1.2652477e-03, 1.2621434e-03, 1.2561048e-03,
       1.2535236e-03, 1.2495258e-03, 1.2459648e-03, 1.2432419e-03,
       1.2422416e-03, 1.2344334e-03, 1.2312152e-03, 1.2290000e-03,
       1.2242110e-03, 1.2212174e-03, 1.2175622e-03, 1.2160403e-03,
       1.2136658e-03, 1.2076140e-03, 1.2045379e-03, 1.2030181e-03,
       1.1989798e-03, 1.1968840e-03, 1.1942502e-03, 1.1930905e-03,
       1.1908448e-03, 1.1838028e-03, 1.1835219e-03, 1.1779685e-03,
       1.1746801e-03, 1.1728758e-03, 1.1686299e-03, 1.1665858e-03,
       1.1643243e-03, 1.1604020e-03, 1.1552661e-03, 1.1529251e-03,
       1.1510504e-03, 1.1470491e-03, 1.1460664e-03, 1.1438235e-03,
       1.1409966e-03, 1.1387529e-03, 1.1371174e-03, 1.1327459e-03,
       1.1287875e-03, 1.1264761e-03, 1.1226551e-03, 1.1196607e-03,
       1.1154821e-03, 1.1145177e-03, 1.1075584e-03, 1.1058630e-03,
       1.1012587e-03, 1.1001255e-03, 1.0975766e-03, 1.0972272e-03,
       1.0951937e-03, 1.0914726e-03, 1.0900162e-03, 1.0861802e-03,
       1.0840353e-03, 1.0830464e-03, 1.0811270e-03, 1.0777710e-03,
       1.0755782e-03, 1.0691205e-03, 1.0672041e-03, 1.0635090e-03,
       1.0617800e-03, 1.0609117e-03, 1.0571751e-03, 1.0546790e-03,
       1.0533786e-03, 1.0514263e-03, 1.0500281e-03, 1.0476074e-03,
       1.0432813e-03, 1.0409728e-03, 1.0375225e-03, 1.0368954e-03,
       1.0343557e-03, 1.0332770e-03, 1.0305531e-03, 1.0290233e-03,
       1.0245773e-03, 1.0212926e-03, 1.0188833e-03, 1.0158732e-03,
       1.0129571e-03, 1.0096993e-03, 1.0088251e-03, 1.0072937e-03,
       1.0036835e-03, 1.0023044e-03, 1.0013669e-03, 9.9791889e-04,
       9.9393202e-04, 9.9041429e-04, 9.8958844e-04, 9.8861451e-04,
       9.8543265e-04, 9.8320865e-04, 9.8036614e-04, 9.7900617e-04,
       9.7519025e-04, 9.7234739e-04, 9.7157952e-04, 9.6790551e-04,
       9.6686563e-04, 9.6319115e-04, 9.6008438e-04, 9.5921673e-04,
       9.5648045e-04, 9.5345109e-04, 9.4933936e-04, 9.4838301e-04,
       9.4600307e-04, 9.4302365e-04, 9.4043498e-04, 9.4030437e-04,
       9.3636307e-04, 9.3427306e-04, 9.3210937e-04, 9.3088619e-04,
       9.2812913e-04, 9.2574290e-04, 9.2285121e-04, 9.2165195e-04,
       9.2041236e-04, 9.1480190e-04, 9.1369177e-04, 9.1036223e-04,
       9.0827199e-04, 9.0613757e-04, 9.0295123e-04, 9.0134458e-04,
       8.9969387e-04, 8.9733157e-04, 8.9331629e-04, 8.9099573e-04,
       8.9061272e-04, 8.8774116e-04, 8.8659150e-04, 8.8437548e-04,
       8.8050991e-04, 8.7922381e-04, 8.7588484e-04, 8.7355875e-04,
       8.7249145e-04, 8.7162643e-04, 8.6706202e-04, 8.6498278e-04,
       8.6225587e-04, 8.6142303e-04, 8.5793971e-04, 8.5635908e-04,
       8.5406663e-04, 8.5158937e-04, 8.4997498e-04, 8.4691262e-04,
       8.4407255e-04, 8.4299617e-04, 8.3928474e-04, 8.3767000e-04,
       8.3487103e-04, 8.3178765e-04, 8.2918518e-04, 8.2712906e-04,
       8.2564267e-04, 8.2237856e-04, 8.2021224e-04, 8.1944047e-04,
       8.1401371e-04, 8.1189093e-04, 8.0952351e-04, 8.0669700e-04,
       8.0116687e-04, 8.0072938e-04, 7.9862331e-04, 7.9565437e-04,
       7.9355080e-04, 7.8953465e-04, 7.8892265e-04, 7.8466174e-04,
       7.8022201e-04, 7.7700464e-04, 7.7279302e-04, 7.7202613e-04,
       7.6655636e-04, 7.6456275e-04, 7.6151220e-04, 7.5811159e-04,
       7.4963906e-04, 7.4570614e-04, 7.4139284e-04, 7.3910563e-04,
       7.3581678e-04, 7.2715903e-04, 7.2551757e-04, 7.1492733e-04,
       7.0619286e-04, 6.8900490e-04, 6.3116988e-04, 5.5659632e-04,
       5.4640311e-04, 4.6178058e-04, 3.8572747e-04, 2.5240699e-04,
       1.8324256e-04, 1.5303958e-04, 4.2368989e-13], dtype=float32)
```

In [140]:
```
type(pca.explained_variance_ratio_)
```

Out[140]:
```
numpy.ndarray
```

In [ ]:
```
pca.explained_variance
```

In [143]:
```
sorted_components = np.sort(pca.explained_variance_ratio_, axis=-1, kind=None, o
shortened_components=sorted_components[:20]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-143-a8e5af444712> in <module>
----> 1 sorted_components = np.sort(pca.explained_variance_ratio_, axis=-1, kin
d=None, order=None)
      2 shortened_components=sorted_components[:20]

~\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py in sort(a, axis, kind,
 order)
    932     else:
    933         a = asanyarray(a).copy(order="K")
--> 934     a.sort(axis=axis, kind=kind, order=order)
    935     return a
    936

TypeError: expected bytes, NoneType found
```

In [124]:
```
shortened_components
```

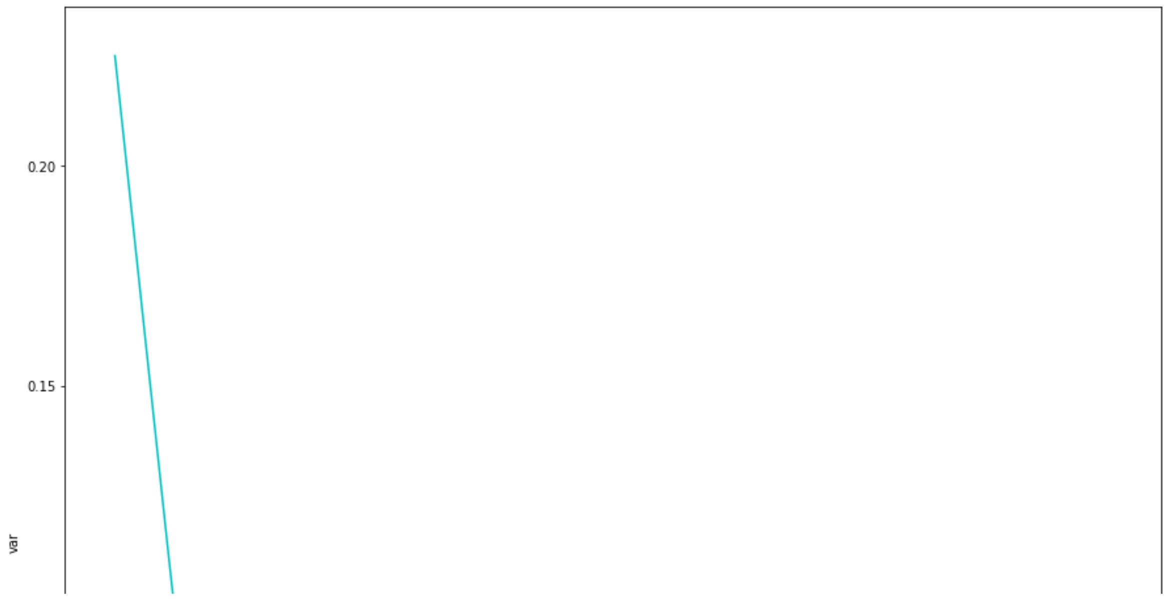
Out[124]:
```
array([0.22507139, 0.11385646, 0.0272289 , 0.01823406, 0.0138927 ,
       0.01038271, 0.00970589, 0.0074357 , 0.00624059, 0.00552639,
       0.00517382, 0.0050704 , 0.00452513, 0.00435713, 0.00424785,
       0.00397213, 0.00386279, 0.0038356 , 0.00375894, 0.00366369],
      dtype=float32)
```

In [127]:
```
mylist = []
for i in range(20):
    mylist.append("PCA%i" % i)
mylist
```

Out[127]:
```
['PCA0',
 'PCA1',
 'PCA2',
 'PCA3',
 'PCA4',
 'PCA5',
 'PCA6',
 'PCA7',
 'PCA8',
 'PCA9',
 'PCA10',
 'PCA11',
 'PCA12',
 'PCA13',
 'PCA14',
 'PCA15',
 'PCA16',
 'PCA17',
 'PCA18',
 'PCA19']
```

In [149]:
```
#Creating a scree plot with the first 20 components.
plt.figure(figsize=(15,15))
df = pd.DataFrame({'var':shortened_components,
             'PC':mylist})
sns.lineplot(x='PC',y="var",
           data=df, color="c")
```

Out[149]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x12185f6fc50>
```



In [133]:
```
#Well that's very bizarre. These are totally out of order.
```

In [ ]:
```
coef = svr.coef_
# reverse feature selection
coef = feature_selection.inverse_transform(coef)
# reverse variance threshold
coef = variance_threshold.inverse_transform(coef)
# reverse masking
weight_img = nifti_masker.inverse_transform(coef)

# Create the figure
from nilearn.plotting import plot_stat_map, show
bg_filename = gray_matter_map_filenames[0]
z_slice = 0


fig = plt.figure(figsize=(5.5, 7.5), facecolor='k')
# Hard setting vmax to highlight weights more
display = plot_stat_map(weight_img, bg_img=bg_filename,
                        display_mode='z', cut_coords=[z_slice],
                        figure=fig, vmax=1)
display.title('SVM weights', y=1.2)

# Measure accuracy with cross validation
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(anova_svr, gm_maps_masked, age)

# Return the corresponding mean prediction accuracy
prediction_accuracy = np.mean(cv_scores)
print("=== ANOVA ===")
print("Prediction accuracy: %f" % prediction_accuracy)
print("")

### Inference with massively univariate model #################################
print("Massively univariate model")

# Statistical inference
from nilearn.mass_univariate import permuted_ols
data = variance_threshold.fit_transform(gm_maps_masked)
neg_log_pvals, t_scores_original_data, _ = permuted_ols(
    age, data,  # + intercept as a covariate by default
    n_perm=2000,  # 1,000 in the interest of time; 10000 would be better
    n_jobs=1)  # can be changed to use more CPUs
signed_neg_log_pvals = neg_log_pvals * np.sign(t_scores_original_data)
signed_neg_log_pvals_unmasked = nifti_masker.inverse_transform(
    variance_threshold.inverse_transform(signed_neg_log_pvals))

# Show results
threshold = -np.log10(0.1)  # 10% corrected

fig = plt.figure(figsize=(5.5, 7.5), facecolor='k')

display = plot_stat_map(signed_neg_log_pvals_unmasked, bg_img=bg_filename,
                        threshold=threshold, cmap=plt.cm.RdBu_r,
                        display_mode='z', cut_coords=[z_slice],
                        figure=fig)
title = ('Negative $\log_{10}$ p-values'
         '\n(Non-parametric + max-type correction)')
```

```
display.title(title, y=1.2)

n_detections = (signed_neg_log_pvals_unmasked.get_data() > threshold).sum()
print('\n%d detections' % n_detections)

show()
```

In [69]:

```
#All k-fold cross-validation stuff here
from sklearn.model_selection import train_test_split
```