实验 3：fMRI 数据处理分析基础

2024 年 5 月 7 日

实验 3：fMRI 数据处理分析基础

1 实验简介

本实验通过开源工具与开源数据集，探索 fMRI 大脑功能成像数据的组织、查看与分析。

通过本实验，你将通过学习操纵真实脑影像数据，掌握以下技能：

- 了解 fMRI 数据的国际通用标准；
- 掌握如何使用 Python 及开源工具包 Nilearn 进行医学影像查看、处理；
- 如何应用机器学习技术构建和评估脑影像的分类模型。

2 实验准备

首先，你需要安装完成本实验所需的工具包。为了加快安装速度，请确保 `pip` 包管理器已切换到国内源。本实验的依赖包列举如下：

- `Nilearn`: 用于查看和处理 fMRI 脑影像数据，特别是建立脑成像相关的机器学习模型；
- `pandas`: 面向表格数据分析的 Python 库；
- `Scikit-Learn`: Python 机器学习库。

[1]:
```
import pkgutil

# 检查 pandas 库是否已安装
if pkgutil.find_loader('pandas'):
    print('Pandas is available.')
else:
    !pip install pandas -i https://pypi.tuna.tsinghua.edu.cn/simple/
```

```
# 检查 Scikit-Learn 库是否已安装
if pkgutil.find_loader('sklearn'):
    print('Scikit-Learn is available.')
else:
    !pip install sklearn -i https://pypi.tuna.tsinghua.edu.cn/simple/

# 检查 Nileran 库是否已安装
if pkgutil.find_loader('nilearn'):
    print('Nilearn is available.')
else:
    !pip install nilearn -i https://pypi.tuna.tsinghua.edu.cn/simple/
```

```
Pandas is available.
Scikit-Learn is available.
Nilearn is available.
```

[2]:
```
# 忽略 Warning 输出
import warnings
warnings.filterwarnings("ignore")
```

3 初识脑影像数据

我们使用 Nilearn 自带的 MNI152 数据样例，学习 Nifti 文件的加载和简单处理。

实验报告要点：- 在实验报告中，请详细记录实验过程，包括代码及运行结果；- 除实验过程外，在报告的适当位置加入你对以下问题的理解（请结合本实验手册，并查阅第三方资料）：1. 什么是 Nifti 标准？Nifti 格式的数据是怎样组织神经影像数据的？2. 什么是神经影像数据标准模板？什么是 MNI-ICBM152 模板？MNI-ICMB152 模板对大脑坐标系是怎样规定的？3. 在脑影像数据分析过程中，为什么需要对成像数据进行平滑处理？

[3]:
```
# 导入 MNI152 数据的路径
from nilearn.datasets import MNI152_FILE_PATH

# MNI152_FILE_PATH 是一个文件路径，首先不妨查看这个位置
print(f"Path to MNI152 template: {MNI152_FILE_PATH}")
```

```
Path to MNI152 template: C:\Users\Lenovo\anaconda3\lib\site-
packages\nilearn\datasets\data\mni_icbm152_t1_tal_nlin_sym_09a_converted.nii.gz
```

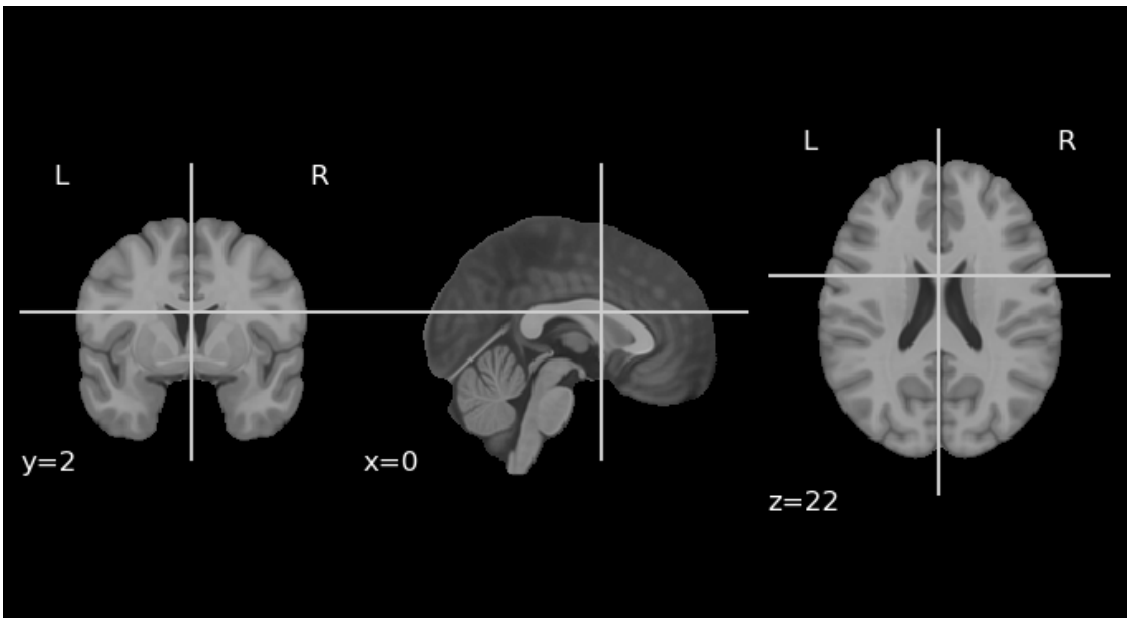
> Nifti（Neuroimaging Informatics Technology Initiative，即神经影像信息学技术倡议）是神经影像数据的一套完整的标准，旨在加强神经影像领域的数据共享和学术合作。通过以下链接进一步了解 Nifti 相关信息。Nifti - Neuroimaging Informatics Technology Initiative

3.1 查看数据

Nilearn 提供了 plotting 模块,用于查看神经影像数据。例如,我们使用 `plot_anat` 函数查看 MNI152 样例数据。

[4]:
```
from nilearn import plotting


plotting.plot_anat(MNI152_FILE_PATH)
```

[4]:
```
<nilearn.plotting.displays._slicers.OrthoSlicer at 0x1e980057970>
```



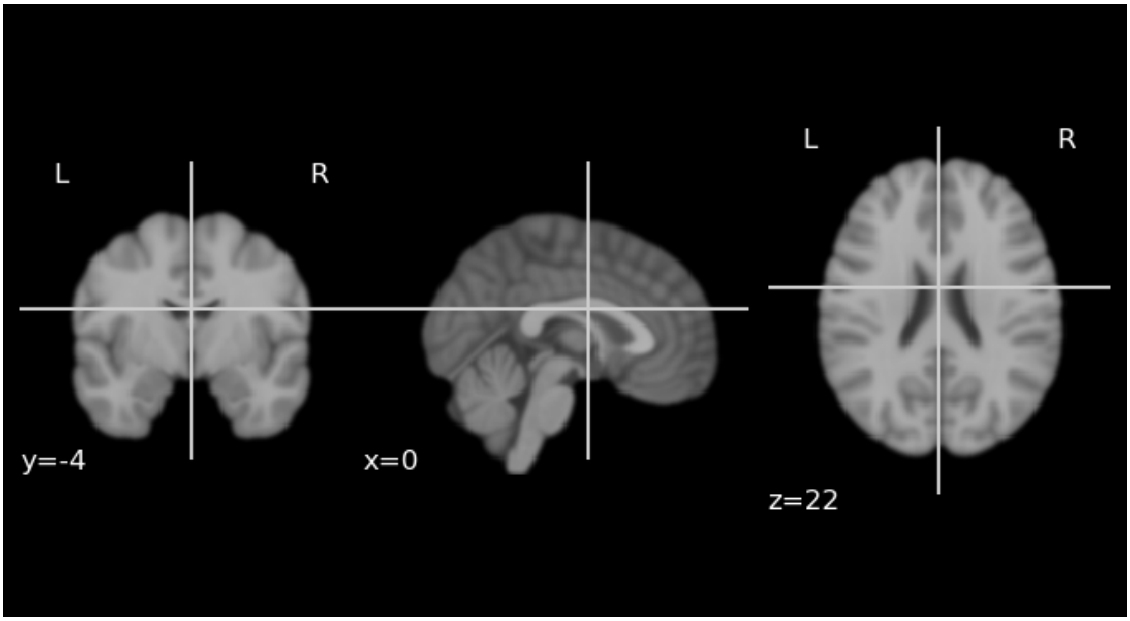
> plotting 模块包含了丰富的可视化功能函数，通过以下链接了解它们。Plotting brain images

3.2 数据处理

尝试对数据进行平滑（smoothing）处理。我们将使用nilearn.image.smooth_img函数。

[5]:
```
from nilearn import image

# smooth_img 函数返回一个 Nifti1Image 类型的内存对象
smooth_anat_img = image.smooth_img(MNI152_FILE_PATH, fwhm=3)

# 查看平滑后的图像
plotting.plot_anat(smooth_anat_img)
```

[5]:
```
<nilearn.plotting.displays._slicers.OrthoSlicer at 0x1e98021bd30>
```

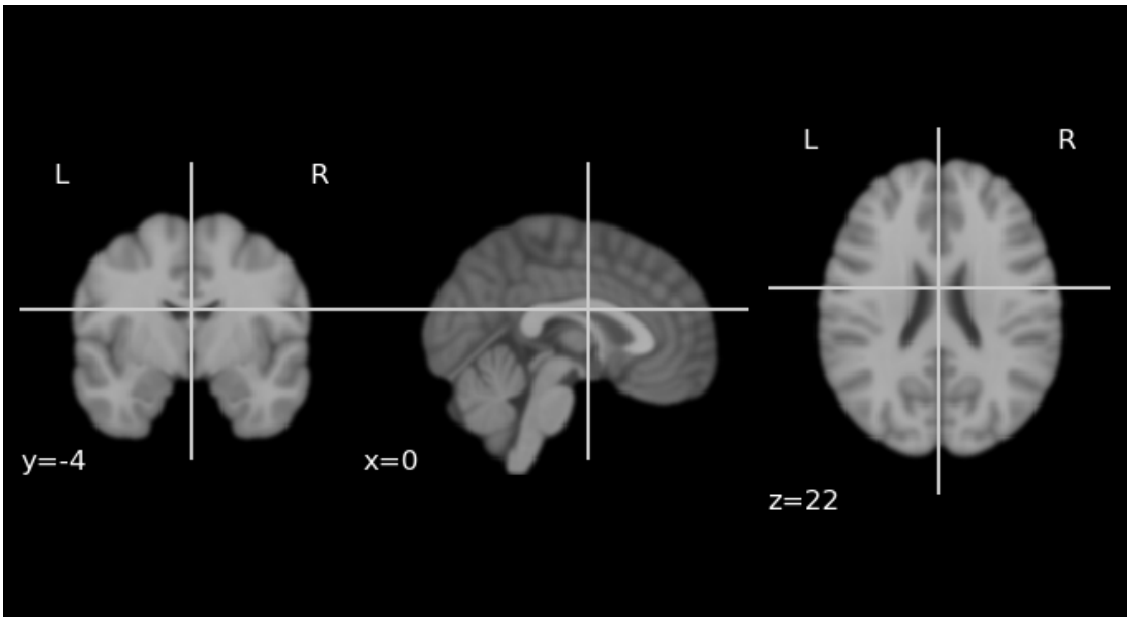


3.3 处理结果的存储和加载

[6]:
```
# 保存处理结果到文件
smooth_anat_img.to_filename("smooth_anat_img.nii.gz")
```

[7]:
```
# 重新加载并绘制
plotting.plot_anat("smooth_anat_img.nii.gz")
```

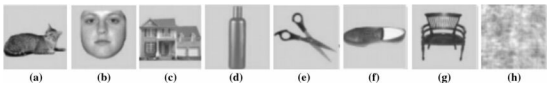
[7]: <nilearn.plotting.displays._slicers.OrthoSlicer at 0x1e98034b7f0>



4 构建脑影像处理机器学习模型

在实验中，我们将基于 Haxby 数据集构建一个机器学习分类模型，该模型在输入脑成像数据，输出预测结果（受试者当前注视的是人类面孔还是猫的图片）。

Haxby 数据集是一个 4-D 时间序列 fMRI 数据集，来自于人脸和物体表征的研究。在 Haxby 的研究中，包含 6 名被试，每名被试完成 12 组实验。在每组实验中，让被试观看 8 种物体类别的灰度图像，每组以 24 秒的休息时间间隔开来。每张图像呈现 500ms，紧接着是 1500ms 的刺激间隔。全脑 fMRI 数据以 2.5s 的像重复时间记录，一种刺激的 block 约含 9 个像，每张图像包含 40x64x64 个体素。



> J.V. Haxby et al. “Distributed and Overlapping Representations of Faces and Objects in Ventral Temporal Cortex”, Science vol 293 (2001), p 2425.-2430. Link

实验报告要点：

- 在实验报告中，请详细记录实验过程，包括代码及运行结果；

- 除实验过程外，在报告的适当位置加入你对以下问题的理解（请结合本实验手册，并查阅第三方资料）：
  1. 简要介绍 Haxby 数据集及与之关联的认知神经科学研究；
  2. 如何查看 4D fMRI 数据？
  3. 构建脑影像分类机器学习预测模型需要注意哪些问题？
  4. 针对脑影像分类模型，你能想到哪些应用场景？

4.1 下载 Haxby 数据集

使用 `nilearn.datasets.fetch_haxby` 函数可以方便地将 Haxby 研究数据集下载到本地磁盘。

如果下载速度慢，可以通过以下链接下载文件，并解压到`%homepath%/nilearn_data` 目录下。

数据下载链接：[Link] （访问码：4qmr）

[8]:
```
from nilearn import datasets

# Haxby 数据集包含 6 名被试，在不指定下载哪一组的情况下，将默认下载第 2 名被试
# 如果数据已经下载过，那么将直接加载数据
haxby_dataset = datasets.fetch_haxby()

# 查看该数据集在本地的存储位置
print(f"Datasets are stored in: {datasets.get_data_dirs()}")
```

```
Datasets are stored in: ['C:\\Users\\Lenovo/nilearn_data']
```

[9]:
```
# 'func'是一组 nifti 文件名，这里取第一个文件
fmri_filename = haxby_dataset.func[0]

# 查看该文件的存放位置
print(f"First subject functional nifti images (4D) are at: {fmri_filename}")
```

```
First subject functional nifti images (4D) are at:
C:\Users\Lenovo/nilearn_data\haxby2001\subj2\bold.nii.gz
```

4.2 数据查看

由于数据是 4D 的（每个时刻包含很多 3D EPI 影像），我们无法直接使用 `nilearn.plotting.plot_anat` 函数进行可视化。

这里我们通过 `nilearn.image.mean_img` 计算得到一张对不同时刻的影像进行 “平均” 的影像。

Tip：`view_img` 函数得到的结果是交互式的，可以用鼠标拖拽蓝色十字，查看不同位置的脑影像。

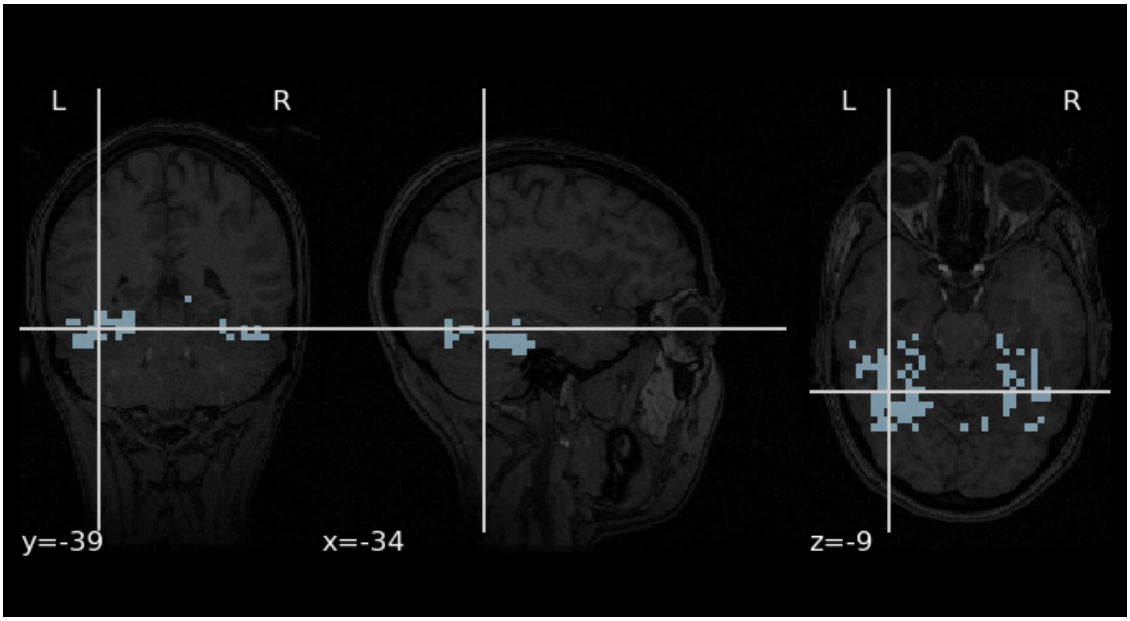
[10]:
```
from nilearn.image import mean_img

plotting.view_img(mean_img(fmri_filename), threshold=None)
```

[10]:
```
<nilearn.plotting.html_stat_map.StatMapView at 0x1e9804627c0>
```

Haxby 数据集还提供了一个腹侧颞叶皮层（Ventral temporal cortex）的 mask。通过 `plot_roi` 查看 mask 的位置。

[11]:
```
mask_filename = haxby_dataset.mask_vt[0]

# 可视化掩膜，也就是将原始图像作为背景，叠加 mask 区域
plotting.plot_roi(mask_filename, bg_img=haxby_dataset.anat[0], cmap="Paired")
```

[11]:
```
<nilearn.plotting.displays._slicers.OrthoSlicer at 0x1e98048ac70>
```



查看数据标签，也就是每个时刻被试者的行为状态。

[12]:
```
import pandas as pd

# 加载被试的行为信息
behavioral = pd.read_csv(haxby_dataset.session_target[0], delimiter=" ")
print(behavioral)
```

```
     labels  chunks
0      rest       0
1      rest       0
2      rest       0
3      rest       0
4      rest       0
…       …       …
1447   rest      11
1448   rest      11
1449   rest      11
1450   rest      11
1451   rest      11

[1452 rows x 2 columns]
```

Haxby 数据集关联的任务是视觉识别任务，因此，每个标签记录了当前被试者看到的图片的内容。通过以下代码查看标签的分类方法。

[13]:
```
conditions = behavioral["labels"]

# 去除重复记录
print(conditions.unique())
```

```
['rest' 'scissors' 'face' 'cat' 'shoe' 'house' 'scrambledpix' 'bottle'
 'chair']
```

标签共包括 8 种图片内容，或者 “rest” 状态。简洁起见，我们只关注 “face” 和 “cat” 两个场景。我们通过以下代码对数据进行过滤，仅保留 “face” 和 “cat” 有关的数据。

[14]:
```
# 创建过滤条件掩膜
condition_mask = conditions.isin(["face", "cat"])
```

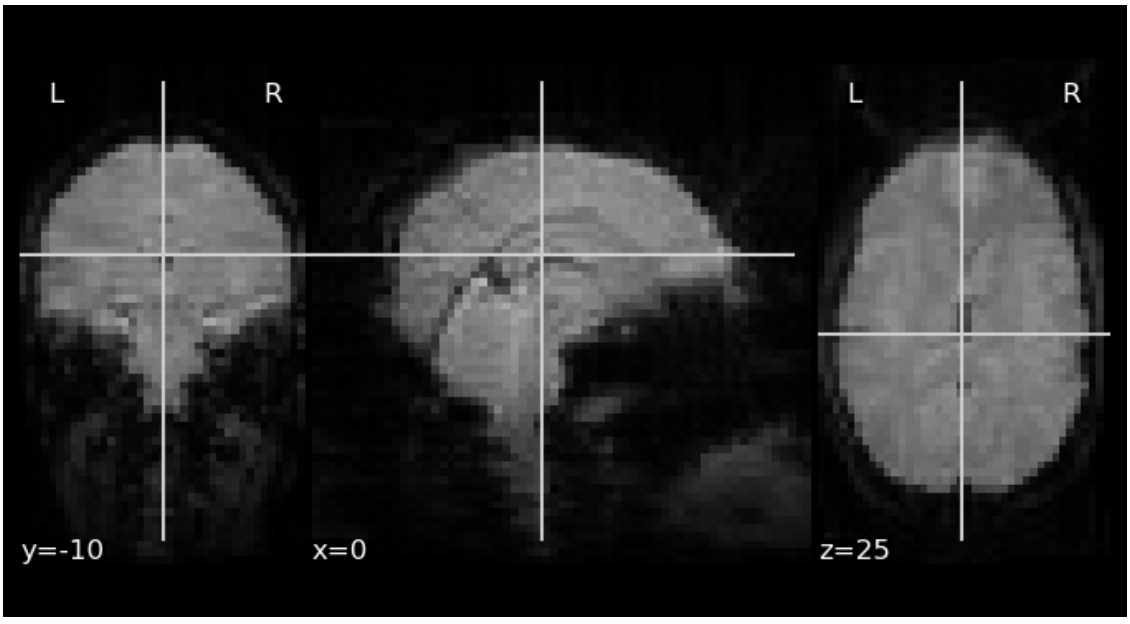
```
# 对标签进行过滤
conditions = conditions[condition_mask]

# 转换为 numpy 数组
conditions = conditions.values

# 查看数据长度
print(f"Label shape: {conditions.shape}.")

# 使用 index_img 对影像进行过滤
from nilearn.image import index_img
fmri_niimgs = index_img(fmri_filename, condition_mask)
print(f"Imageset shape: {fmri_niimgs.shape}.")
```

```
Label shape: (216,).
Imageset shape: (40, 64, 64, 216).
```

观察可知，筛选后的数据包含 216 个时刻的数据。如果要查看某一特定时刻的影像，只需使用 index_img 筛选即可。

[15]:
```
# 查看某时刻的影像
plotting.plot_anat(index_img(fmri_niimgs, 215))
```

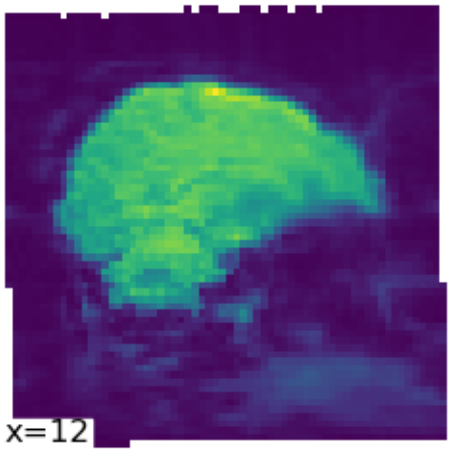
[15]:
```
<nilearn.plotting.displays._slicers.OrthoSlicer at 0x1e9805c4670>
```

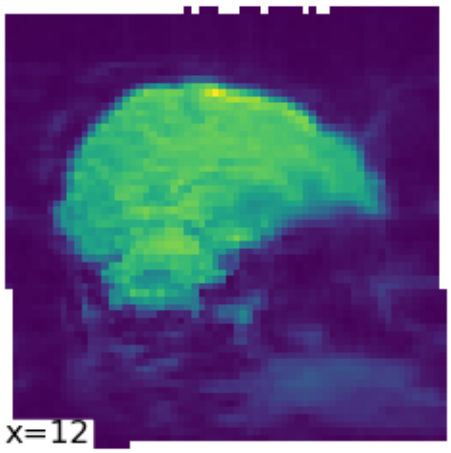


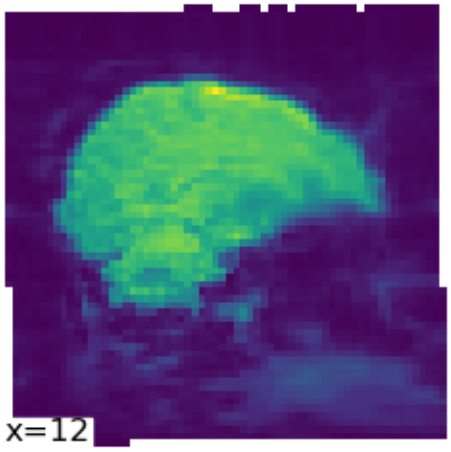
此外，还可以通过使用 iter_img 函数，查看连续的一组影像。

[16]:
```
from nilearn.image import iter_img

fmri_niimgs_subset = index_img(fmri_niimgs, slice(0, 3))
print(fmri_niimgs_subset.shape)

for img in iter_img(fmri_niimgs_subset):
    plotting.plot_img(
        img, threshold=3, display_mode="x", cut_coords=1, colorbar=False
    )
```

```
(40, 64, 64, 3)
```







4.3 尝试创建模型

我们使用支持向量机（SVM）训练一个简单的模型，模型输入为 3D 影像（VT 皮质掩膜区域），输出为被试看到的物体。

首先定义模型。

[17]:
```
from nilearn.decoding import Decoder

decoder = Decoder(
```

```
    estimator="svc", mask=mask_filename, standardize="zscore_sample"
)
```

使用所有数据进行拟合。

[18]:
```
decoder.fit(fmri_niimgs, conditions)
```

用训练好的模型进行预测。

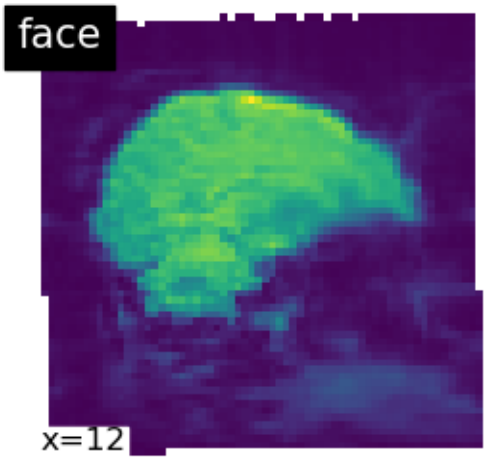
[19]:
```
prediction = decoder.predict(fmri_niimgs)
print(prediction)
```

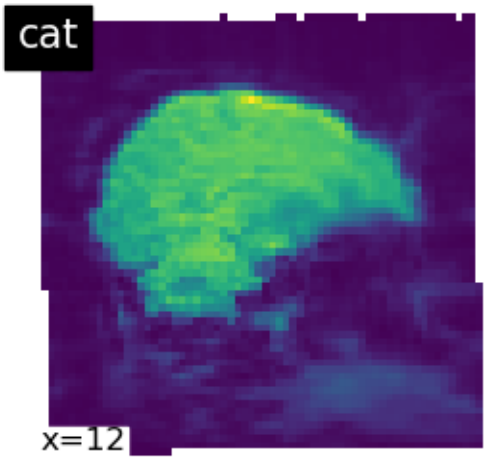
```
['face' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'cat'
 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'face' 'face' 'face'
 'face' 'face' 'face' 'face' 'face' 'face' 'cat' 'cat' 'cat' 'cat' 'cat'
 'cat' 'cat' 'cat' 'cat' 'face' 'face' 'face' 'face' 'face' 'face' 'face'
 'face' 'face' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat'
 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'face' 'face' 'face'
 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face'
 'face' 'face' 'face' 'face' 'face' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat'
 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat'
 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face'
 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'cat' 'cat' 'cat'
 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'face' 'face' 'face' 'face' 'face'
 'face' 'face' 'face' 'face' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat'
 'cat' 'cat' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face'
 'face' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'face'
 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'cat' 'cat' 'cat'
 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat'
 'cat' 'cat' 'cat' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face'
 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face' 'face'
 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat' 'cat']
```

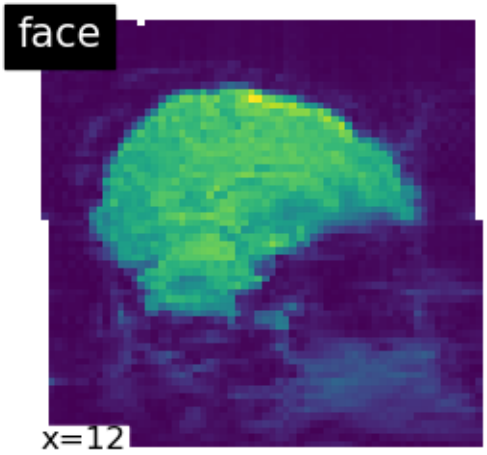
为了更加直观，我们还可以查看部分影像及预测结果样例。

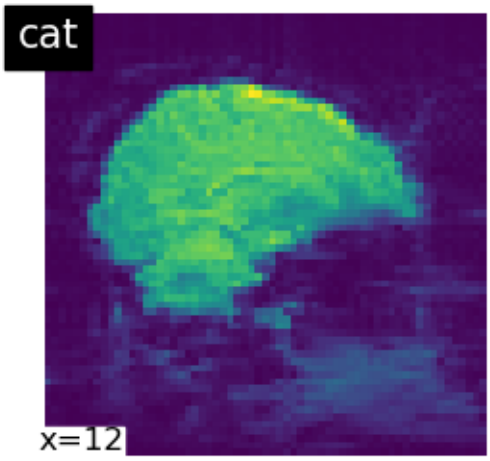
[20]:
```
selected_indices = [0, 9, 20, 32]
fmri_niimgs_subset = index_img(fmri_niimgs, selected_indices)

for idx, img in enumerate(iter_img(fmri_niimgs_subset)):
```

```
plotting.plot_img(
    img, threshold=3, display_mode="x", cut_coords=1, colorbar=False,
    title=prediction[selected_indices[idx]]
)
```









4.4 通过交叉验证方法评估模型效果

在一个标准的机器学习模型构建过程中，我们需要划分训练集、测试集，仅使用训练集数据进行训练，且仅在测试集上评估模型效果。

这里，我们使用最后 30 个 data point 作为测试集。

[21]:
```
# 划分训练集、测试集
fmri_niimgs_train = index_img(fmri_niimgs, slice(0, -30))
fmri_niimgs_test = index_img(fmri_niimgs, slice(-30, None))
```

```
conditions_train = conditions[:-30]
conditions_test = conditions[-30:]

# 定义模型
decoder = Decoder(
    estimator="svc", mask=mask_filename, standardize="zscore_sample"
)

# 训练模型
decoder.fit(fmri_niimgs_train, conditions_train)

# 在测试集上获得预测结果
prediction = decoder.predict(fmri_niimgs_test)

# 在测试集上评估模型效果
predicton_accuracy = (prediction == conditions_test).sum() / float(
    len(conditions_test)
)
print(f"Prediction Accuracy: {predicton_accuracy:.3f}")
```

```
Prediction Accuracy: 0.767
```

与一般机器学习模型的构建一样，我们还可以借助 scikit-learn 包，使用 K 折交叉验证方法评估模型效果。

[22]:
```
from sklearn.model_selection import KFold
import warnings
warnings.filterwarnings("ignore")

cv = KFold(n_splits=5)

for fold, (train, test) in enumerate(cv.split(conditions), start=1):
    decoder = Decoder(
        estimator="svc", mask=mask_filename, standardize="zscore_sample"
    )
    decoder.fit(index_img(fmri_niimgs, train), conditions[train])
    prediction = decoder.predict(index_img(fmri_niimgs, test))
```

```
    predicton_accuracy = (prediction == conditions[test]).sum() / float(
        len(conditions[test])
    )
    print(
        f"CV Fold {fold:01d} | "
        f"Prediction Accuracy: {predicton_accuracy:.3f}"
    )
```

```
CV Fold 1 | Prediction Accuracy: 0.886
CV Fold 2 | Prediction Accuracy: 0.767
CV Fold 3 | Prediction Accuracy: 0.767
CV Fold 4 | Prediction Accuracy: 0.698
CV Fold 5 | Prediction Accuracy: 0.744
```

Nilearn 包的 Decoder 也提供了 K 折交叉验证功能。

[23]:
```
n_folds = 5
decoder = Decoder(
    estimator="svc",
    mask=mask_filename,
    standardize="zscore_sample",
    cv=n_folds,
    scoring="accuracy",
)
decoder.fit(fmri_niimgs, conditions)
```

[24]:
```
# 检查每个 Fold 的性能指标
print(decoder.cv_params_["face"])
```

```
{'C': [100.0, 100.0, 100.0, 100.0, 100.0]}
```

还可使用“留一法”，即在训练时每次留出一组作为验证集，所有其他组的数据作为训练集。

[25]:
```
# 只选择“face”、“cat”的实验
session_label = behavioral["chunks"][condition_mask]
```

[26]:
```
from sklearn.model_selection import LeaveOneGroupOut

cv = LeaveOneGroupOut()
```

```
decoder = Decoder(
    estimator="svc", mask=mask_filename, standardize="zscore_sample", cv=cv
)
decoder.fit(fmri_niimgs, conditions, groups=session_label)

print(decoder.cv_scores_)
```

```
{'cat': [1.0, 1.0, 1.0, 1.0, 0.9629629629629629, 0.8518518518518519,
0.9753086419753086, 0.40740740740740744, 0.9876543209876543, 1.0,
0.9259259259259259, 0.8765432098765432], 'face': [1.0, 1.0, 1.0, 1.0,
0.9629629629629629, 0.8518518518518519, 0.9753086419753086, 0.40740740740740744,
0.9876543209876543, 1.0, 0.9259259259259259, 0.8765432098765432]}
```

4.5 查看模型权重

通过 `decoder.coef_` 可查看 SVM 模型的权重。每个体素一个系数。

[27]:
```
coef_ = decoder.coef_
print(coef_)
```

```
[[-3.89376915e-02 -1.87166827e-02 -3.23027223e-02 -2.88747334e-02
   4.18696787e-02  1.10743982e-02  1.69998374e-02 -5.50954056e-02
  -1.94204165e-02 -3.51227347e-02  1.08511681e-02 -1.28796909e-02
  -1.54677084e-02 -3.78908998e-02 -3.69167840e-02  2.28087889e-02
   6.56416964e-03 -7.65753762e-03  1.67105832e-02 -8.02137755e-03
   5.29515654e-02 -8.17594607e-02 -6.36992178e-02  2.41325756e-02
   4.59874916e-02 -2.22603142e-02 -1.77309364e-02  2.22197290e-02
  -9.53184412e-03  5.76047658e-02  2.14300211e-02 -9.14227736e-02
   4.03660748e-03 -2.89276009e-02 -3.89030724e-02 -3.35114396e-02
   2.21396058e-03  8.73132450e-03 -3.37416625e-02 -2.41275823e-02
  -6.81648703e-02  1.65406612e-02  2.70783862e-02 -6.56852899e-03
  -1.21663769e-02  5.47674473e-02  8.13285814e-03  3.60955646e-02
  -1.52764432e-02  7.02913565e-02  1.28090839e-03  2.08007984e-02
  -4.09926781e-03  3.72429614e-02 -3.77393519e-02 -1.03858769e-02
  -2.38237998e-02 -5.48880033e-02  4.43026847e-02 -1.47419267e-01
  -2.34042309e-02  1.87113551e-02  6.65858173e-02 -9.07602948e-02
```

```
-1.22033551e-02 -2.95655393e-03  3.22091719e-02 -3.04054082e-02
 6.15345375e-02  1.12248738e-02  1.93774715e-02 -1.30541742e-02
 4.42975695e-02 -2.23066221e-02  6.88147064e-02  1.69390082e-02
 1.78947266e-02  1.00274909e-02  2.99186443e-02 -2.52170794e-02
 1.06153452e-02 -6.31955274e-03  2.21515214e-03 -2.23349123e-02
 1.42561009e-02 -1.53123664e-02 -1.98226768e-02 -4.32637932e-02
-4.55125554e-02  3.41589273e-02 -2.79199202e-02 -2.80909099e-02
-3.70158923e-02 -5.71450744e-02 -6.98948421e-02  3.20175436e-03
-8.35451135e-03 -3.37628232e-02  3.04261597e-02  8.68457113e-03
 6.19382450e-03  5.94180006e-02  9.07296304e-03 -1.48931682e-02
 1.43558956e-02 -1.09026977e-02  2.67697872e-02  4.73787818e-02
-2.96432090e-02  3.09423446e-02  1.57928945e-02 -3.16720148e-02
-4.00105412e-02 -5.40261333e-02  2.82612250e-02 -1.12100968e-02
-5.45402703e-02  6.32178200e-02 -1.49997205e-02  2.47548040e-03
-4.56644074e-02 -1.83882880e-02  1.19956465e-02 -3.72172435e-02
-2.25526962e-03  4.58656637e-02  4.79167954e-02  2.51823329e-03
-4.31721930e-02 -5.35345059e-03  5.76994251e-02  7.40847734e-03
-3.20590355e-02  4.35700267e-03  1.68302995e-02 -2.92570098e-02
-2.24505356e-03 -8.30211317e-03 -1.00013090e-02  2.17135896e-02
-1.92615231e-03 -1.33222423e-02 -2.80300840e-02 -1.75293895e-02
-9.17792832e-03 -7.09944207e-03 -1.43033609e-02  5.06831589e-02
-1.84813733e-02 -4.71506499e-02  1.72569072e-02 -4.76642278e-02
-9.09082151e-04  4.00768230e-02  7.53995738e-02  7.25618380e-03
 4.82604971e-02  4.50555181e-02  3.61201998e-02 -8.16481561e-03
 1.95405729e-02  3.57884042e-02  4.89303914e-02  3.82974429e-02
 6.23918887e-02  6.13675345e-02 -1.68752151e-02  1.66514280e-02
 3.35521731e-02 -1.80212942e-02  4.46411139e-02 -3.53245630e-02
-3.67292034e-02 -4.62245046e-03  4.86829780e-02  3.39666640e-02
 6.21723539e-03  1.73612235e-02  2.01697221e-02  2.17099156e-02
 2.91413337e-02  2.37777037e-02  4.84698180e-02 -9.22623940e-03
-2.82637691e-02 -2.13781216e-02  1.80785589e-03  4.79687364e-02
-9.78894516e-03  1.11431567e-02 -1.65019268e-02 -2.89090540e-02
 2.42851412e-02 -1.22347676e-02 -2.92870970e-02 -2.89844022e-02
-3.39533340e-02 -3.65273849e-03  2.65325084e-02  4.58043863e-02
-5.93380514e-02 -2.13630286e-02 -3.09407689e-02  5.50178119e-02
-3.38816368e-02  6.12602888e-03  1.41484344e-02  1.10216703e-02
 5.33811524e-02 -2.12340029e-02  6.37421188e-03 -1.13075798e-02
```

```
-2.64225268e-02 -2.22398642e-02 -5.31921030e-02 -3.98652927e-02
-1.29727842e-01 -3.28092354e-02 -2.89712247e-02 -9.13473174e-03
-7.28723040e-03 -3.71051949e-02 -6.34907364e-02  2.04368630e-03
-8.26794696e-02 -6.71214943e-02 -2.29131089e-03 -2.33452316e-02
 1.77914417e-02 -8.74666874e-02 -2.76477862e-03 -4.38275354e-02
-1.28050956e-02  2.78034482e-02 -4.32695443e-02 -3.22688208e-02
-2.28029175e-02 -2.57414954e-02  2.03623850e-02 -9.90248198e-03
-3.15033664e-02 -1.81419569e-02 -1.12323718e-03 -4.17432985e-02
-6.23480161e-02  2.54492346e-04 -6.73680025e-02  6.53967009e-02
 1.06521268e-02  2.21983349e-02 -1.98728680e-02 -1.85520162e-02
 4.05701932e-02 -3.02838533e-02 -8.10050496e-02 -7.42458176e-02
-4.93847900e-02 -1.01771912e-02  1.09407158e-02 -4.49254936e-02
 2.92747930e-02  7.05309652e-03  5.07529402e-03 -4.84051936e-03
 2.48754471e-03  3.00654713e-02 -2.63087317e-03  4.64687399e-03
 7.90209621e-02  1.04857765e-02  1.68079723e-02 -4.36718179e-02
-1.08855713e-02  2.10241898e-02 -4.41965139e-02  3.16494772e-03
 6.98670572e-02  8.61629778e-02  4.96235851e-02  6.03891153e-03
 5.56494429e-02 -2.98919021e-02  4.13036569e-03 -3.21951487e-02
-3.14990555e-02 -5.31275834e-02  2.67256465e-02  3.14428557e-02
 6.67091111e-03 -1.28703844e-02  2.20151254e-02  5.68523768e-02
 2.25603808e-02 -2.04616769e-02  5.10341950e-03  2.85357769e-02
-1.81663939e-02 -8.48429180e-03 -3.18826007e-02 -1.18498055e-02
-4.10845952e-02  3.11776360e-02  9.63475013e-03 -8.25913490e-03
-3.12226452e-02  8.57656063e-03 -9.70188381e-03  1.32374474e-02
 4.06446748e-02  8.23407462e-03 -3.27355028e-02 -4.33883630e-03
-1.75531114e-02  6.88841984e-03  3.45133333e-02  7.03298300e-02
 2.16784891e-02  5.32225090e-03  8.17563743e-02  6.40062509e-02
-2.31146803e-03 -1.17558583e-02  1.75889236e-01  3.18130066e-02
-3.15886490e-02  3.34028506e-02  2.22781191e-02  1.00231906e-02
-4.74913187e-02 -2.12756820e-02 -3.98718146e-02 -6.04069059e-02
-4.65060680e-02  1.03003436e-02 -3.05702104e-04  1.80743515e-02
-1.75451611e-02 -8.72589671e-02  1.00662699e-01  4.46116235e-03
 7.46870359e-02 -6.13411920e-02  2.81703945e-02 -1.40977833e-02
 3.14638582e-02 -1.63834126e-02  3.66532599e-02 -5.15683696e-03
 1.45093982e-02  6.35867176e-02  2.34598261e-02  8.81062253e-02
 6.15344559e-02 -1.39361834e-02  2.07247387e-02 -3.15452992e-03
 5.15425652e-02 -2.88767380e-02  1.60261837e-02  2.09702337e-02
```

```
-3.29172210e-02 -2.59461542e-02 -5.60400313e-02 -3.64627741e-02
 1.12882214e-02  2.17268096e-02 -1.51638791e-02 -7.82886880e-03
 2.42549933e-02  9.47013399e-02 -2.63031861e-02  1.17356365e-04
-5.24171416e-03  4.17988173e-02  8.85681117e-02  6.23651762e-03
 1.86601326e-02  1.54628572e-02  3.50535682e-03  6.20599761e-03
-1.19791134e-02  1.59526512e-02  7.12130414e-03 -8.93192540e-02
-3.54344991e-03  1.23477920e-02  3.03927754e-02 -2.37295213e-02
-3.82793016e-02 -4.98744631e-02  4.66896569e-02 -1.23292872e-02
-1.10332327e-02  2.18105706e-02  2.18719709e-02  2.63539399e-02
 1.05280045e-02  1.84618294e-02  8.36147869e-04 -6.65210147e-03
 3.49398287e-02  1.49352490e-02 -1.11599400e-02  6.69104640e-03
-2.00058820e-02 -3.99014826e-02  3.01871987e-02 -1.09866549e-02
-4.11780212e-02  2.72052371e-02  1.16425822e-02 -1.55501768e-02
 3.27701512e-02  3.95494538e-02  8.48733894e-03  2.19934951e-02
-9.88666567e-03 -3.61421575e-02 -4.77021466e-02  1.90075033e-02
-5.58286184e-02 -3.31739220e-02 -2.24913558e-02 -3.36175149e-02
-4.07355653e-02  1.08861029e-02  1.12810802e-02  7.63143076e-02
 4.04804847e-03  3.07013522e-02  2.89177565e-02  4.71608101e-03
 5.13384386e-02 -4.10364403e-02  1.23261689e-03 -2.50403775e-02
 5.85904527e-02 -1.04965591e-01 -4.41705064e-02  1.18520106e-02
-5.83203740e-02 -4.82245081e-02  9.17655348e-03  1.03259931e-02
-5.09188228e-03 -3.23390267e-02 -3.19386319e-02 -1.53770002e-02
-5.21212048e-02  1.55619953e-02  2.93484165e-02 -1.92528427e-02
 1.76693981e-02  2.67992145e-02  5.76553944e-02 -1.38163005e-02
 2.60399658e-02  1.50399507e-02  1.27424092e-02 -2.29244152e-02
-1.06665236e-02  9.81937753e-03 -4.77511697e-02  1.64243752e-02]]
```

[28]:
```
print(coef_.shape)
```

```
(1, 464)
```

筛选出 “face” 类的系数。

[29]:
```
coef_img = decoder.coef_img_["face"]
```

将模型权重保存到文件。

[30]:
```
decoder.coef_img_["face"].to_filename("haxby_svc_weights.nii.gz")
```

使用脑影像作为背景，将模型权重进行可视化。

[31]:
```
plotting.view_img(
    decoder.coef_img_["face"],
    bg_img=haxby_dataset.anat[0],
    title="SVM weights",
    dim=-1,
)
```

[31]:
```
<nilearn.plotting.html_stat_map.StatMapView at 0x1e9807760d0>
```

4.6 基线模型

一般地，我们可以将训练得到的模型与随机模型进行比较，以展示模型的可用性。

[32]:
```
dummy_decoder = Decoder(
    estimator="dummy_classifier",
    mask=mask_filename,
    cv=cv,
    standardize="zscore_sample",
)
dummy_decoder.fit(fmri_niimgs, conditions, groups=session_label)

# Now, we can compare these scores by simply taking a mean over folds
print(dummy_decoder.cv_scores_)
```

```
{'cat': [0.38888888888888895, 0.38888888888888895, 0.38888888888888895,
0.6111111111111112, 0.38888888888888895, 0.6111111111111112,
0.38888888888888895, 0.38888888888888895, 0.38888888888888895,
0.38888888888888895, 0.6111111111111112, 0.38888888888888895], 'face':
[0.38888888888888895, 0.38888888888888895, 0.38888888888888895,
0.6111111111111112, 0.38888888888888895, 0.6111111111111112,
0.38888888888888895, 0.38888888888888895, 0.38888888888888895,
0.38888888888888895, 0.6111111111111112, 0.38888888888888895]}
```