log_model

April 26, 2023

[1]:
```python
import ants
import glob
import os
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from multiprocessing import Pool
import gc

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

0.1 Read in the data

All data is in one folder, so we need to split the files into white matter and gray matter

[2]:
```python
def get_ordered_files(directory, prefix):

    # Get a list of all files in the directory
    all_files = os.listdir(directory)

    # Filter files that start with prefix
    files = [f for f in all_files if f.startswith(prefix)]

    # Filter again for wm and gm
    gm_files = [f for f in files if f.endswith('-gm.nii.gz')]
    wm_files = [f for f in files if f.endswith('-wm.nii.gz')]

    # Sort the files list
    sorted_gm_files = sorted(gm_files)
    sorted_wm_files = sorted(wm_files)

    return sorted_gm_files, sorted_wm_files
```

[3]:
```
data_path = '/scratch/users/neuroimage/conda/data'
img_path = os.path.join(data_path, 'preprocessed/imgsss')

smt_files_gm, smt_files_wm = get_ordered_files(img_path, "smt")
snmt_files_gm, snmt_files_wm = get_ordered_files(img_path, "snmt")
nsmt_files_gm, nsmt_files_wm = get_ordered_files(img_path, "nsmt")
```

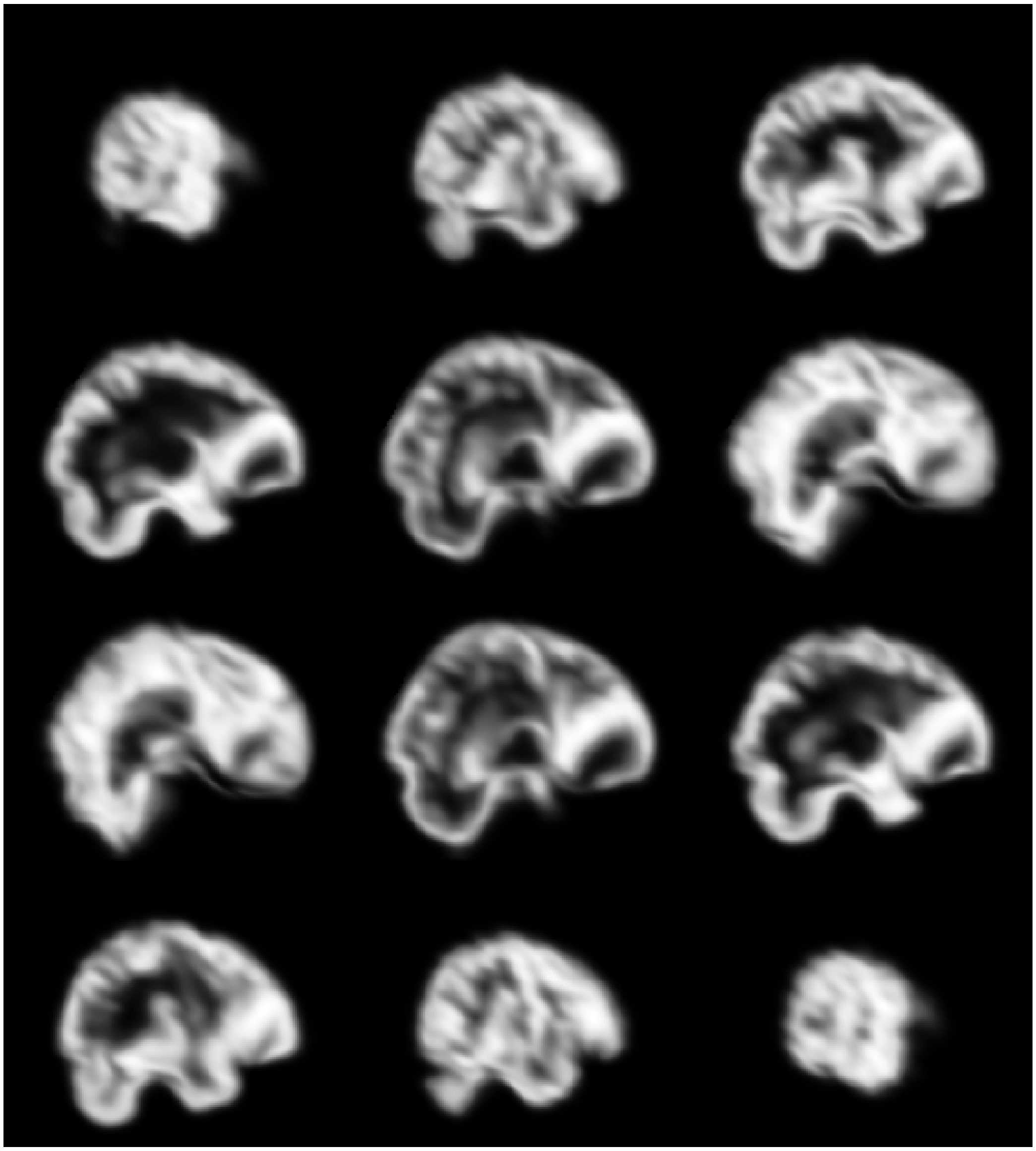
[4]:
```
print(smt_files_gm[0], smt_files_wm[0])
print(snmt_files_gm[0], snmt_files_wm[0])
print(nsmt_files_gm[0], nsmt_files_wm[0])
```

```
smt-002_S_0413-I118675-gm.nii.gz smt-002_S_0413-I118675-wm.nii.gz
snmt-002_S_0413-I118675-gm.nii.gz snmt-002_S_0413-I118675-wm.nii.gz
nsmt-002_S_0413-I118675-gm.nii.gz nsmt-002_S_0413-I118675-wm.nii.gz
```

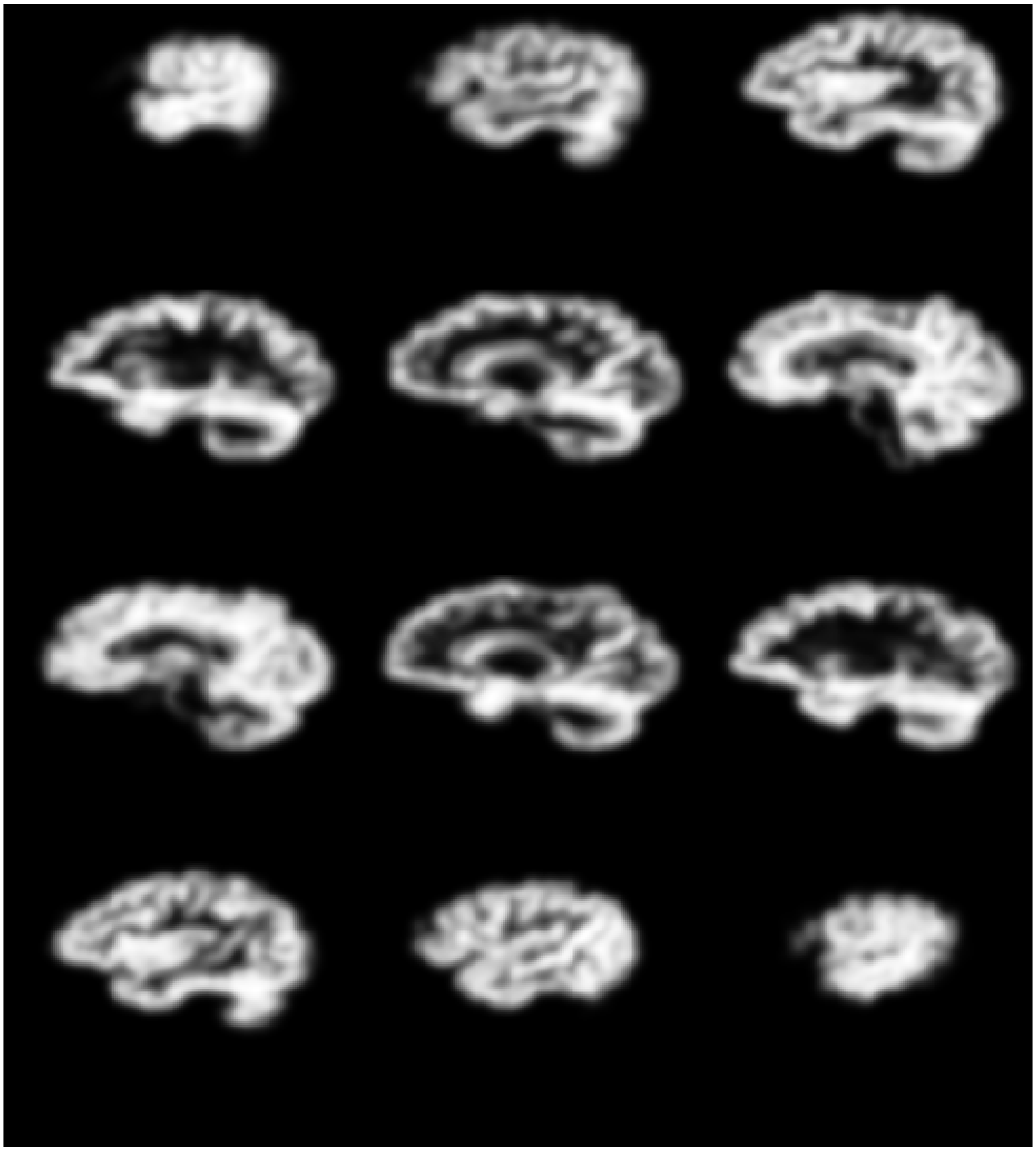
Visualizing the difference in smooth mapped to template, smooth but not mapped to template and not smooth but mappyed to template

[5]:
```
print("SMT")
ants.plot(os.path.join(img_path, smt_files_gm[0]))
print("SNMT")
ants.plot(os.path.join(img_path, snmt_files_gm[0]))
print("NSMT")
ants.plot(os.path.join(img_path, nsmt_files_gm[0]))
```

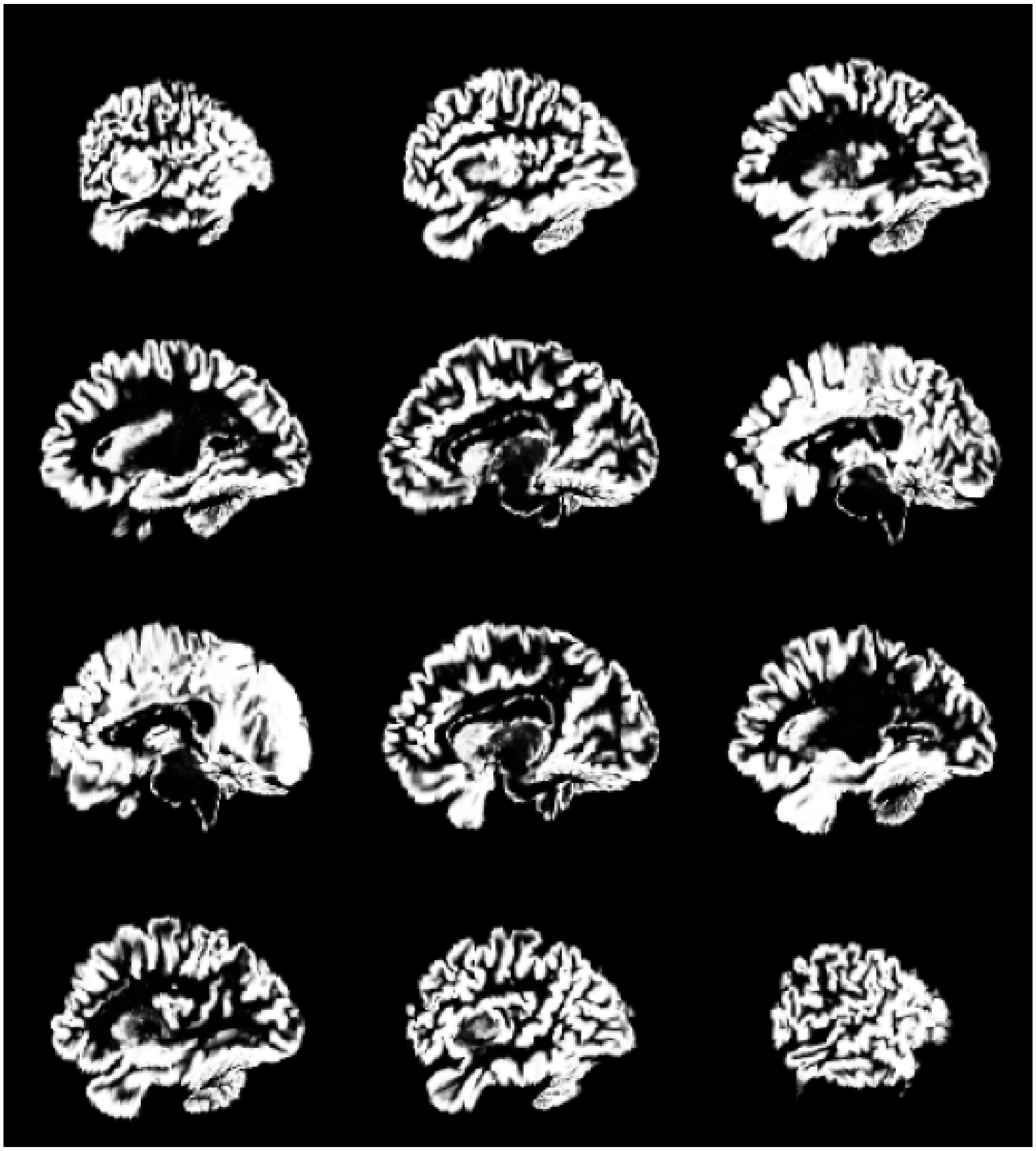
```
SMT
```



SNMT



NSMT



0.2 Create Design Matrix

[6]:
```
def imgs_to_matrix(directory, wm_files=None, gm_files=None, combine=False):
    imgs = []

    if combine:

        for file_grouping in zip(wm_files, gm_files):
            wm_path, gm_path = file_grouping

            wm_img = ants.image_read(os.path.join(directory, wm_path))
```

```python
            gm_img = ants.image_read(os.path.join(directory, gm_path))

            # Clone the white matter image
            comb_img = ants.image_clone(wm_img)

            # Add the gray matter image to the combined image
            comb_img = comb_img + gm_img

            # grab subject and img info
            sub_id_wm, img_id_wm = wm_path.split('-')[1:3]
            sub_id_gm, img_id_gm = gm_path.split('-')[1:3]

            if sub_id_wm != sub_id_gm:
                raise ValueError(f'wm id:{sub_id_wm} ne to gm id:{sub_id_gm}')

            vector =  comb_img.numpy().ravel()

            # turn to matrix, then to 1D array
            img_vec = np.append([sub_id_wm, img_id_wm], vector)
            imgs.append(img_vec)

        X = np.vstack(imgs)
        return X

    else:
        all_files = [wm_files, gm_files]
        both_Xs = []
        for files in all_files:
            imgs = []
            for path in files:
                img = ants.image_read(os.path.join(directory, path))

                # grab subject and img info
                sub_id, img_id = path.split('-')[1:3]

                vector =  img.numpy().ravel()

                # turn to matrix, then to 1D array
                img_vec = np.append([sub_id, img_id], vector)
                imgs.append(img_vec)

            # stack the vectors into a 2D array
            X = np.vstack(imgs)
            both_Xs.append(X)

        # wm_X, gm_X
        return both_Xs
```

[7]:
```python
def matrix_to_df(X):
    # V: Voxel intensity
    # turn matrix to dataframe and name columns
    X_cols = ['Subject', 'Img_ID'] + ['V{}'.format(i+1) for i in range(X.
 ↪shape[1]-2)]
    X_df = pd.DataFrame(X, columns=X_cols)

    return X_df

def clean_data(X_df, md):
    # merge two dfs
    X_cl = md.merge(X_df, on=['Subject', 'Img_ID'])
    X_cl = X_cl.drop(columns=['Img_ID', 'Subject'])

    # create X and y
    y = X_cl['Group'].values
    X = X_cl.drop(columns=['Group'])

    return X, y


def get_metadata(data_path):
    # Clean metadata dataframe
    md = pd.read_csv(os.path.join(data_path, 'ADNI1_Complete_2Yr_3T_4_18_2023.
 ↪csv'))

    md = md.rename(columns={'Image Data ID': 'Img_ID'})

    md = md.drop(columns=['Visit', 'Modality', 'Description', 'Type', 'Acq␣
 ↪Date', 'Format', 'Downloaded'])

    md['Group'] = md['Group'].map({'CN':0, 'MCI':1, 'AD':2})
    md['Sex'] = md['Sex'].map({'F':0, 'M':1})

    return md
```

Testing out individual functions

[8]:
```python
X_wm, X_gm = imgs_to_matrix(img_path, smt_files_gm[0:10], smt_files_wm[0:10],␣
 ↪combine=False)
X_comb = imgs_to_matrix(img_path, smt_files_gm[0:10], smt_files_wm[0:10],␣
 ↪combine=True)
```

[9]:
```python
md = get_metadata(data_path)
md.head()
```

[9]:
```
     Img_ID     Subject  Group  Sex  Age
0  I205567  136_S_1227      1    0   66
1   I66824  136_S_1227      1    0   65
2   I79080  136_S_1227      1    0   65
3  I143856  136_S_1227      1    0   67
4   I99265  136_S_1227      1    0   66
```

[10]:
```
df = matrix_to_df(X_comb)
df.head()
```

[10]:
```
      Subject   Img_ID   V1   V2   V3   V4   V5   V6   V7   V8  … V2122936  \
0  002_S_0413  I118675  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …      0.0
1  002_S_0413  I120746  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …      0.0
2  002_S_0413  I128346  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …      0.0
3  002_S_0413   I40657  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …      0.0
4  002_S_0413   I64551  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …      0.0

  V2122937 V2122938 V2122939 V2122940 V2122941 V2122942 V2122943 V2122944  \
0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
1      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
2      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
3      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
4      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0

  V2122945
0      0.0
1      0.0
2      0.0
3      0.0
4      0.0

[5 rows x 2122947 columns]
```

[11]:
```
X, y = df.pipe(clean_data, md=md)
```

[12]:
```
X.head()
```

[12]:
```
   Sex  Age   V1   V2   V3   V4   V5   V6   V7   V8  … V2122936 V2122937  \
0    1   79  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …      0.0      0.0
1    1   82  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …      0.0      0.0
2    1   81  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …      0.0      0.0
3    1   80  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …      0.0      0.0
4    1   79  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …      0.0      0.0

  V2122938 V2122939 V2122940 V2122941 V2122942 V2122943 V2122944 V2122945
0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
1      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
```

```
2       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0
3       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0
4       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0

[5 rows x 2122947 columns]
```

0.3 PCA and Logistic Regression

[13]:
```
def perform_pca(X, y):
    # Preprocess and scale the data
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Split the data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,␣
↪test_size=0.2, random_state=10)

    # Fit and transform the PCA model on the training set
    pca = PCA(random_state=10)

    X_train_pca = pca.fit_transform(X_train)

    # calculate cumulative variance
    cumulative_variance = np.cumsum(pca.explained_variance_ratio_)

    # find the index where cumulative variance reaches 95%
    n_components = np.argmax(cumulative_variance >= 0.95) + 1
    print(f'n components:{n_components}')

    # re-fit PCA with the chosen number of components
    pca = PCA(n_components=n_components, random_state=10)
    X_train_pca = pca.fit_transform(X_train)

    # Transform the test set using the trained PCA model
    X_test_pca = pca.transform(X_test)

    return X_train_pca, y_train, X_test_pca, y_test


def perform_logreg(X_train_pca, y_train, X_test_pca, y_test):
    clf = LogisticRegression(random_state=10, penalty=None,␣
↪multi_class='multinomial').fit(X_train_pca, y_train)
    y_preds = clf.predict(X_test_pca)

    acc = sum(y_preds == y_test) / len(y_test)
```

```
    return acc


def full_pipeline(X_matrix):

    md = get_metadata(data_path)

    df = matrix_to_df(X_matrix)

    X, y = df.pipe(clean_data, md=md)

    X_train_pca, y_train, X_test_pca, y_test = perform_pca(X, y)

    return X_train_pca, y_train, X_test_pca, y_test
```

0.4 Comparisons

[14]:
```
X_wm, X_gm = imgs_to_matrix(img_path, smt_files_gm, smt_files_wm, combine=False)
X_comb = imgs_to_matrix(img_path, smt_files_gm, smt_files_wm, combine=True)
```

[15]:
```
X_train_wm, y_train_wm, X_test_wm, y_test_wm = full_pipeline(X_wm)
X_train_gm, y_train_gm, X_test_gm, y_test_gm = full_pipeline(X_gm)
X_train_cb, y_train_cb, X_test_cb, y_test_cb = full_pipeline(X_comb)
```

```
n components:146
n components:188
n components:179
```

[16]:
```
logreg_wm_acc = perform_logreg(X_train_wm, y_train_wm, X_test_wm, y_test_wm)
logreg_gm_acc = perform_logreg(X_train_gm, y_train_gm, X_test_gm, y_test_gm)
logreg_cb_acc = perform_logreg(X_train_cb, y_train_cb, X_test_cb, y_test_cb)
```

```
/scratch/users/neuroimage/conda/venv/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
```

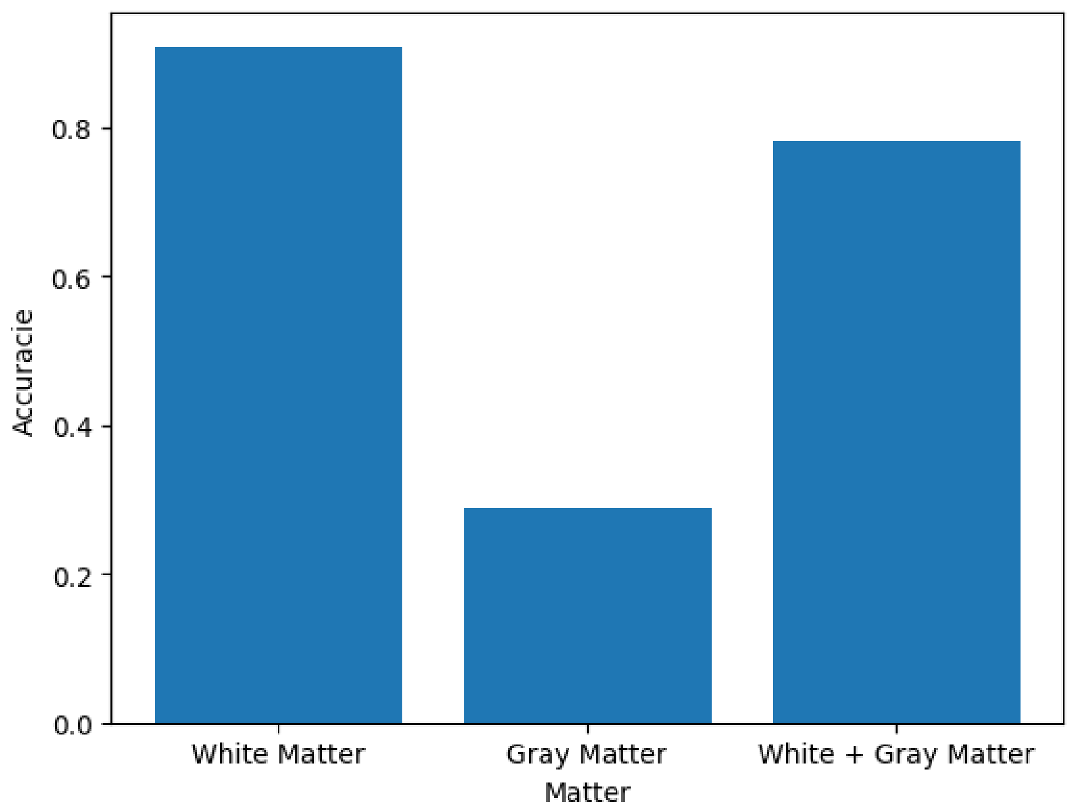
[17]:
```
print(f"Classification accuracy under WM data: {logreg_wm_acc}")
print(f"Classification accuracy under GM data: {logreg_gm_acc}")
```

```python
print(f"Classification accuracy under Combined data: {logreg_cb_acc}")
```

```
Classification accuracy under WM data: 0.9080459770114943
Classification accuracy under GM data: 0.28735632183908044
Classification accuracy under Combined data: 0.7816091954022989
```

[18]:
```python
# Create a bar plot
accuracies = [logreg_wm_acc, logreg_gm_acc, logreg_cb_acc]
labels = ['White Matter', 'Gray Matter', 'White + Gray Matter']
plt.bar(labels, accuracies)

# Add labels and title
plt.xlabel('Matter')
plt.ylabel('Accuracie')
# plt.title('Bar plot of 3 values')
# Show the plot
plt.show()
```



[ ]: