Estimating a cortical thickness map for T1-weighted brain imaging

Introduction

The human cerebral cortex consists of a greatly folded sheet of neurons. The thickness of this sheet varies and is called the cortical thickness. There exist many algorithms and programs that analyze cortical thickness as it is an important area of the brain to study for signs of diseases. The goal of this project is to develop something similar in order to estimate the cortical thickness map of a T1-weighted image, which is a type of brain scan. To do this, we must determine the distance between the gray/white matter surface and the pial surface. The gray/white matter surface is where the gray matter meets the white matter and the pial surface is the outer boundary of the brain's gray matter.

Preparation & Methodology:

Provided files

For this project, we were provided with 3 files:

raw_t1_subject_01.nii.gz *: The raw T1 scan of a brain labeled subject 1*
raw_t1_subject_02.nii.gz *: The raw T1 scan of a brain labeled subject 2*
thickness_map_subject_01.nii.*gz : The cortical thickness map of subject 1 (to be used as a comparison for our result)*

After loading the required libraries, including nibabel (a library for accessing neuroimaging files), all 3 files were loaded into separate variables.

```
sub1 = nib.load('raw_t1_subject_01.nii').get_fdata()
sub2 = nib.load('raw_t1_subject_02.nii').get_fdata()
sub1map = nib.load('thickness_map_subject_01.nii').get_fdata()
```

We next applied a gaussian filter to the subject 1 raw in order to soothe the image and make segmenting white and gray matter easier.

Library used from skimage (Python) `from skimage import filters`

Segmentation of white matter

In order to get the distances needed between the gray/white matter surface and the pial surface, we must first isolate those surfaces. We start with the white/gray matter surface by segmenting the white matter from the rest of the image.

The white matter voxels in the image are naturally of a higher value than the rest of the brain, thus we use a simple threshold to isolate only voxels with high values. We then use a binary dilation morphological operation to expand the isolated white matter voxels. This creates an outer border that we then isolate by subtracting the original white matter segmentation. This provides us with our white/gray matter boundary, as the white matter border is naturally also the border between white and gray matter.

```
wh = sub1f > 70
dwh = morphology.binary_dilation(wh)
ewh = dwh.astype(float) - wh
```

Segmentation of gray matter

We next need to segment the gray matter. This is done very similarly to the white matter. We use a threshold to find all voxels between two values that correspond to the gray matter. Once segmented, we make sure to assign the float type to the segmentation as our distance calculation will be done with float values. As before, we apply a binary dilation operation and subtract the initial gray matter segmentation from the dilated one. This results in the gray matter boundary. Since we need the pial surface and not the whole gray matter boundary, we must perform some more operations.

```
gr = (sub1f < 70) & (sub1f > 45)
gr = gr.astype(float)
dgr = morphology.binary_dilation(gr)
egr = dgr.astype(float) - gr
```

Finding the pial surface

Now that we have both the white/gray matter surface and the gray matter boundary, we can isolate the pial surface. To do this we perform another binary dilation on the white/gray matter surface and then subtract that dilation from the gray matter boundary. The result is only the gray matter outer boundary. However, since we subtracted two already segmented images, some negative values pollute our result. To get rid of them, we simply isolate all values that are above 0. This results in a clean pial surface segmentation.

```
dwh2 = morphology.binary_dilation(ewh)
dgr2 = egr.astype(float) - dwh2
fegr = dgr2 > 0
finalbound = ewh + fegr
```

Initializing coordinate arrays

In order to find the distance between the white/gray matter surface and the pial surface, we first need to initialize arrays of coordinates for the white/gray matter surface, the pial surface, and the full gray matter segmentation (which will be useful for the final step). To do this, we use the **argwhere** function in NumPy to generate arrays of non-zero coordinates for all three segmentations.

```
no_zero_whb = np.argwhere(ewh == 1)
no_zero_grb = np.argwhere(fegr == 1)
no_zero_whb.shape
```

```
no_zero_grf = np.argwhere(gr == 1)
no_zero_grf.shape
```

Determining cortical distance

Now that we have both surfaces, we can calculate the cortical distance. To do this we use the Euclidean formula for distance calculation:

$$d(x,\ y)\ = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

We create a function to calculate the Euclidean distance that takes in variables x and y and returns the distance value.

We then use a for loop to iterate through all non-zero white/gray matter boundary voxels and find all distances to every pial surface voxel. We then find the minimum distance value from the list of white/gray voxel distances, which indicates the distance to the closest pial voxel. We then assign that distance value to the corresponding voxel on the original white/gray matter boundary segmentation.

```
def euclid(x,y):
    return np.sqrt(np.sum((x-y)**2, axis=1))

for x in range(no_zero_whb.shape[0]):
    d = euclid(no_zero_whb[x,:], no_zero_grb)
    mind = np.min(d)
    ewh[no_zero_whb[x,0],no_zero_whb[x,1],no_zero_whb[x,2]] = mind
```

Getting the cortical thickness map

Now that we have assigned the distance values to the white/gray voxels, we can use them to assign the distance to all other voxels in the gray matter segmentation and create the thickness map. To do this, we first define a function to find the closest white/gray voxel to each gray matter voxel. We used the KDTree data structure for this purpose.

Library used from scipy (Python) `from scipy import spatial`

We then, as in the previous step, used a for loop to iterate through every gray matter voxel, find its closest white/gray matter border neighbor, and assign the value associated with the neighbor to the gray matter voxel. This created a cortical thickness map with each gray matter voxel having a distance value corresponding to its cortical thickness.

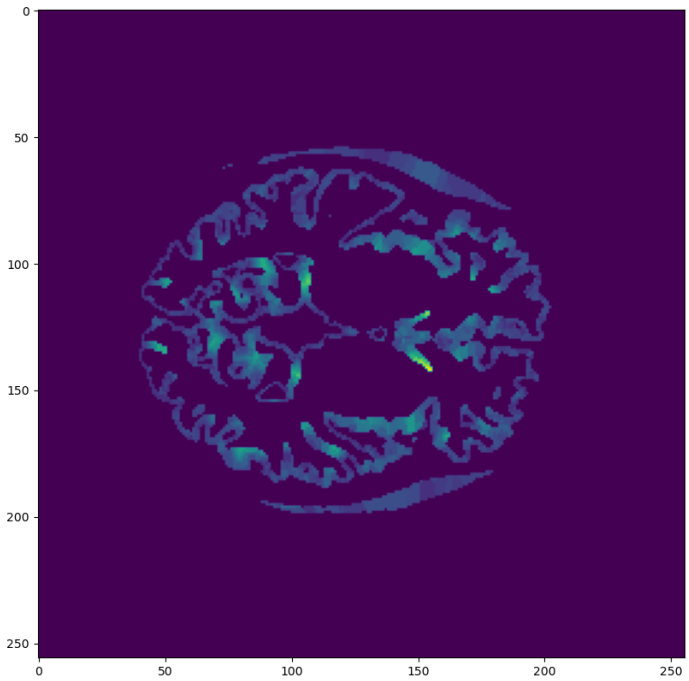
```
def kdt(i,j):
    return spatial.KDTree(i).query(j)[1]

for x in range(no_zero_grf.shape[0]):
    pt = no_zero_grf[x,:]
    cl_dist = no_zero_whb[kdt(no_zero_whb,pt)]
    gr[pt[0],pt[1],pt[2]] = ewh[cl_dist[0],cl_dist[1],cl_dist[2]]
```

Results:

We started by running our program with the subject 1 raw file first as we had the thickness map already provided and we could easily compare them and make changes.

Here is our resulting cortical thickness map for the middle slice of the subject 1 file:



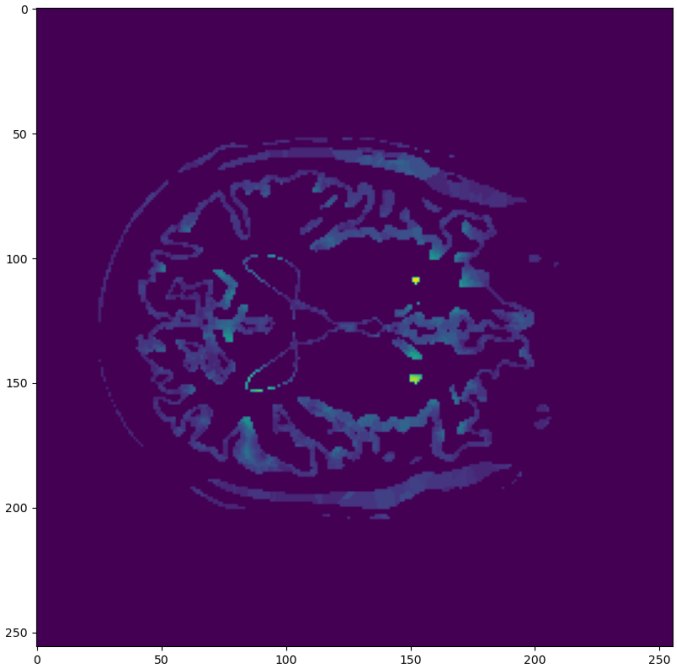
We can see that the areas where the green is darker are where the cortical distance is smaller and the areas that are lighter correspond to thicker gray matter.

We can also notice some imperfections, such as a section in the middle right of the image where the thickness appears small visually, but the color/distance value is assigned as one of the highest in the image.

There are also extra features on the top and bottom of the brain. These exist because, during our gray matter threshold segmentation, this section were also included.

When comparing our result with the given thickness map for subject 1, it is apparent that further optimization in the segmentation process would have been necessary to match it more accurately.

We then also ran our program with the subject 2 raw file which produced a similar result, albeit with a different shape:



We can observe that with this subject, there are more superficial features around the gray matter as well as a more connected inner section. The same flaws found with the first subject are also found here. This could also be a result of us using a different method of distance calculation than used for the provided thickness map.

Challenges

We encountered a few challenges when developing this program.

Initially, we did not know how to proceed after the segmentation of both surfaces. We experimented with several ideas for calculating the distance between the surfaces such as one that involved vectors being cast from one surface to the other and recording the coordinates of each voxel it passed through in order to assign the distance value once found.
This method ended up being too long and complicated to implement and was abandoned.

Another significant challenge was the time that it took to compute a result from the last for a loop. It took several hours and we experimented with filters and boundaries to shorten the processing time but to no avail.

There was also a lot of trial and error in trying to perfect the segmentation of the white and gray matter as well as the surfaces. We tried various thresholds, morphological operations, and filters in order to remove the inter and outer features that were produced. However, we did not find a solution that would at once remove them and also provide an accurate segmentation of the white and gray matter.