Name: Luke Trujillo
Due Date: 3/220

143 Hw # 4

Problem 1. Find two recent ML papers (from 2016 to now) and the corresponding GitHub code which use at least one of the following: (Riemannian) connections, parallel transport, covariant derivatives, and geodesics on manifold. Then study one of the papers in details and make the code running to generate figures in the paper.

Solution: The two papers I decided to look at are:

- Visual Domain Adaptation with Manifold Embedded Distribution Alignment.
  Github: `https://github.com/jindongwang/transferlearning`
- A Plug&Play P300 BCI Using Information Geometry
  Github: `https://github.com/alexandrebarachant/pyRiemann`

I first looked at the first paper listed above, but an issue arises with their code upon running. In a python file, it calls "import bob.linear" but there does not exist such a python module. "Bob" consists of many python modules, but "linear" isn't one of them. I think in the past it did exist, but the bob.linear module has been split up into four different submodules. Since I couldn't figure out which one the authors needed, I looked at the second paper.

The second paper is extremely interesting as it uses information geometry to study a dataset of ERPs (event related potentials), electrical data from the brain, of participants engaging in a brain-computer interface (BCI). Specifically, the participants were playing a newly invented BCI game. The goal of the paper was to come up with a way to classify the ERPs, such as when a player made a mistake, and moreover create a more general BCI which quickly adapts and understands the players. Usually, BCIs require a calibration phase, which is time consuming and inefficient. The researchers wanted to see if they could create a useful BCI which didn't utilize this approach, as they again argue it is time consuming and inefficient.

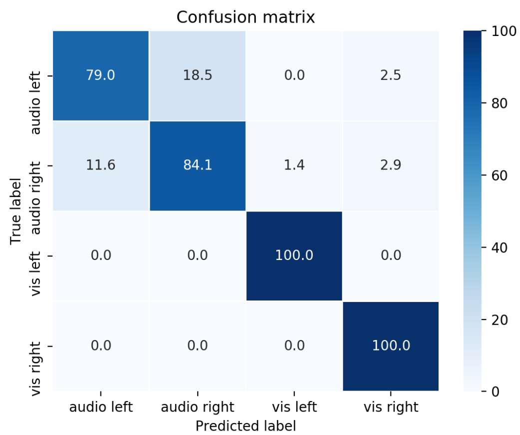
The paper does this by computing geodesics on the manifold of covariance matrices. They arrive at this by interpreting ERP runs as covariance matrices. To compute the geodesics, they first choose an affine invariant distance metric between two covariance matrices σ_1, σ_2

$$D(\sigma_1, \sigma_2) = || \log(\sigma_1^{-1/2} \sigma_2 \sigma_1^{-1/2}) ||_F = \left(\sum_{c=1}^{C} \log^2(\lambda_c) \right)^{\frac{1}{2}}$$

where λ_c are the eigenvalues of $\sigma_1^{-1/2} \sigma_2 \sigma_1^{-1/2}$. This metric offers them "rescaling and normalization of the signals, electrodes permutations, whitening, spatial filtering or source separation" (3) without changing the distance. Ultimately, they decide that this is their optimal metric.

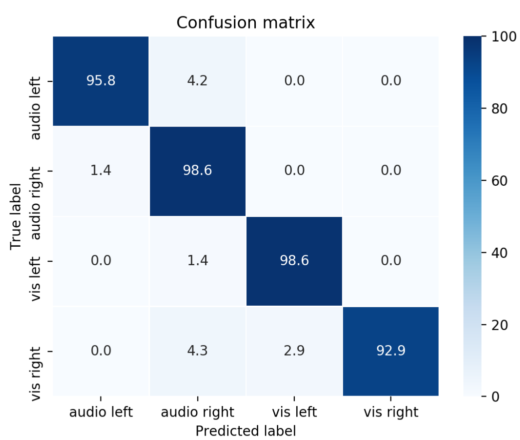
The code attached to this paper is actually a Python module created by the lead author who implemented the methods of the paper. The author left an "Examples" folder for users to test the classification algorithm.

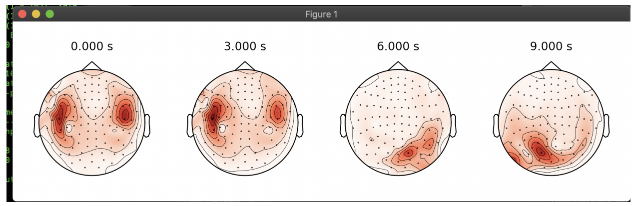
- One of the classification methods that I found in the code involved computing the covariance matrices of the ERPs and then classifying them by their tangent space. Upon running this python script plot_classify_EEG_tangentspace.py, the classification accuracy was 0.902, and the script even returned a confusion matrix.



- Another classification method I found in their code used an MDM (mean distance metric) algorithm, as discussed in their paper. Upon running the python script plot_classify_MEG_mdm.py, the code returned

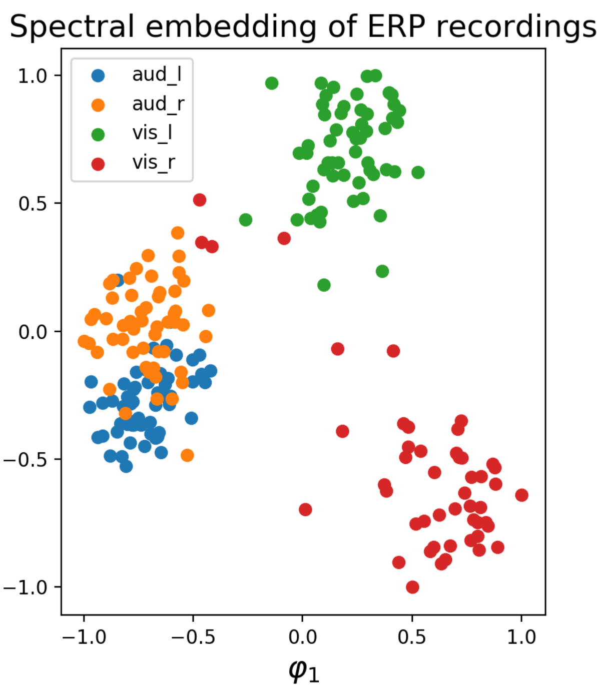
```
Multiclass classification with XDAWN + MDM
              precision    recall  f1-score   support

           1       0.99      0.96      0.97        72
           2       0.91      0.99      0.95        73
           3       0.97      0.99      0.98        73
           4       1.00      0.93      0.96        70

    accuracy                           0.97       288
   macro avg       0.97      0.96      0.97       288
weighted avg       0.97      0.97      0.97       288
```





The above terminal output and confusion matrix reveals the accuracy of their algorithms. I'm not quite sure what the third

- The final script implemented in this example folder includes the plot_embedding_EEG.py, which isn't discussed in their paper (and was probably just added later). This returned a "spectral embedding."



$\square$