NeuroProsthetics

Thijs Lukkien (s3978389) Jiri Derks (s4003039)

March 2024

1.1

In figures 1-4, we highlighted a few areas of noise. In Figure 1 we can see a channel that is picking up predominantly power line noise, with a frequency of 50Hz. Figure 2 depicts some large spikes in the data over several channels, this could be either blinking or a heartbeat. In Figure 3 we see some artifacts across all channels, especially around the time $t = 336$. Figure 4 shows some relatively clean data.

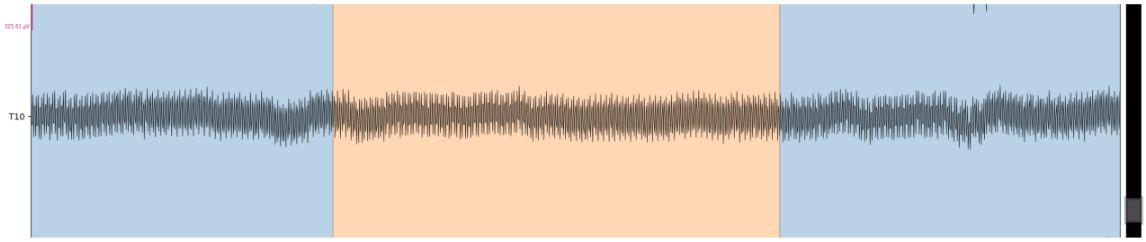

Figure 1: A channel with powerline noise.

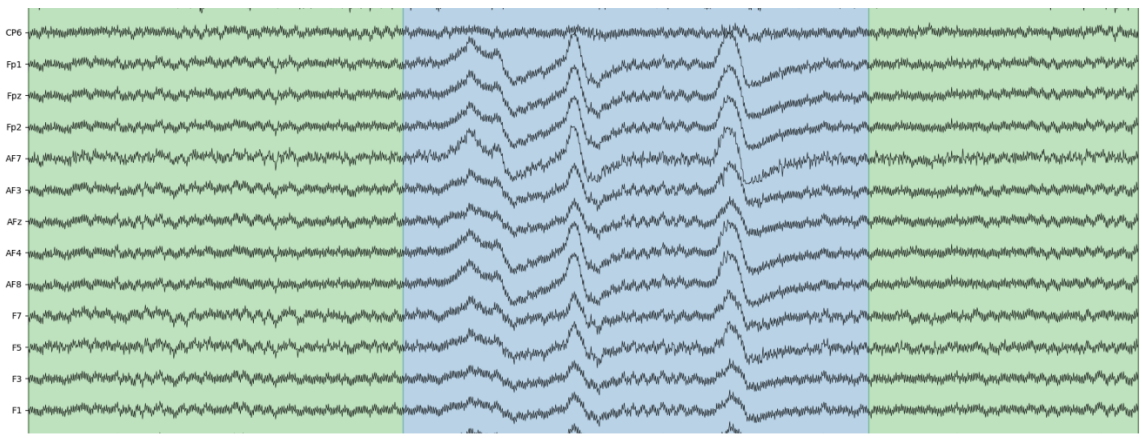

Figure 2: A channel with blinking or heartrate artifacts.

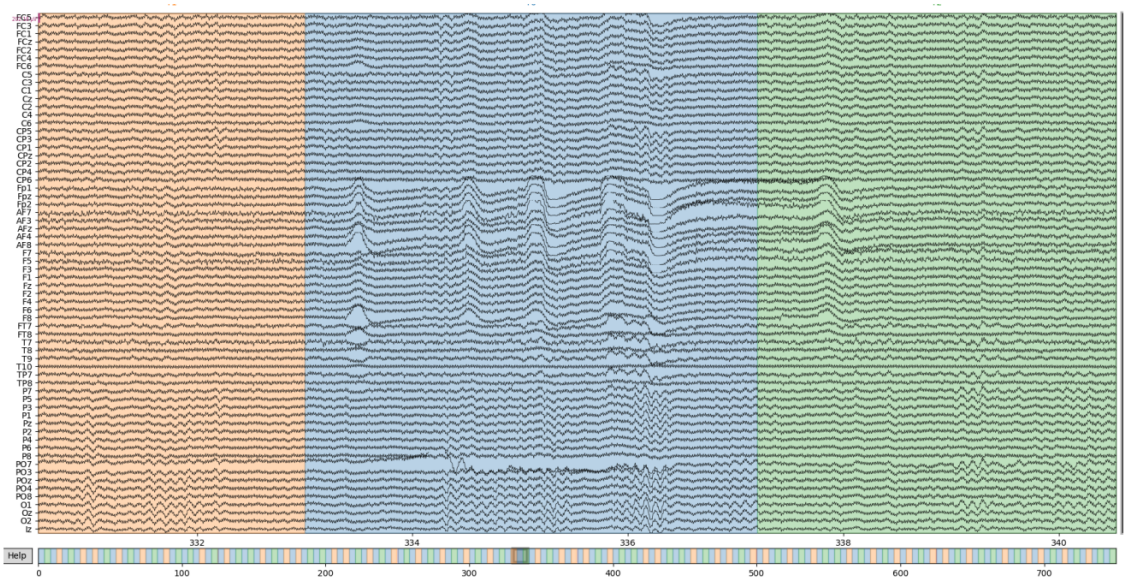

Figure 3: Moments with noise over all channels.

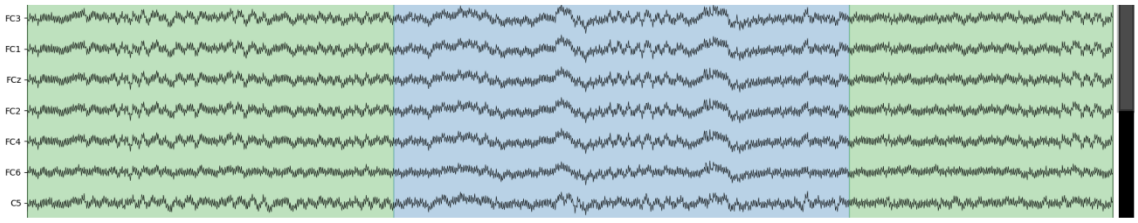

Figure 4: Relatively clean channels.

2a)

Which frequency band did you choose?
The frequency band we chose was 8-30Hz as this encompasses the alpha and beta frequencies which are the frequencies in which we expect to find the motor intention data.

2b)

Which EEG rhythms are kept by this and why do you want to keep those?
From the lecture, we learned that there are several several EEG frequency bands, namely theta (4-8Hz), alpha(8-13Hz), beta(13-30Hz), and gamma(>30Hz). As the data we are interested in (motor imaginary control) is mainly in the alpha and beta bands we want to keep those bands.

2c)

What was the performance of the Machine Learning model before applying the temporal filter, and what is it now?
When performing 5 runs of the machine learning model pipeline, the performance before applying the temporal filter had an average of 0.542. After applying the temporal filter with the low frequency at 8 Hz and the high frequency of 30Hz, the mean over 5 runs was 0.797.

3)

The number of CSP filters currently does not make sense. Make an informed choice of a more appropriate number of CSP filters. Then use csp.plot filters(info=raw.info) to look at the learned CSP filters.

CSP filters aim to differentiate classes by their spatial features. Since each filter can differentiate two classes from one another, it makes sense to use a multiple of the number of classes. Thus, we will need to use a multiple of 2 for the number of CSP components. We tried 2 and 4 components and found that 4 CSP components performed better.

What do you observe? Are there any filters that correspond to the expected activation in the Motor Cortex? Please add an image of the learned CSP filters.

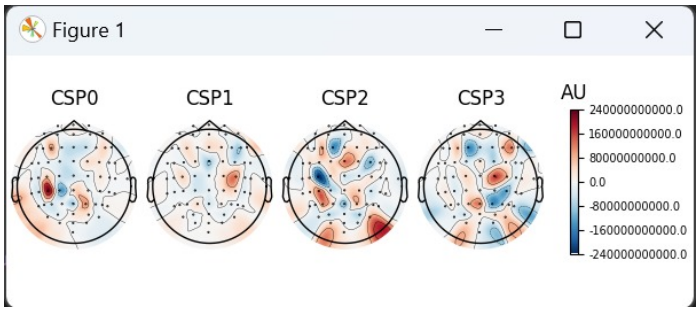

Figure 5: EEG data per CSP component.

In Figure 5 we can see that there are activations in the left and right regions of the brain, these areas correspond to the motor cortex areas of the left and right hand. This is especially clear in *CSP0* and *CSP1*.

4)

Which ranges of hyperparameter values did you cover and why did you choose those? Please add the 3 plots and describe what you discovered in each plot. What are the optimal hyperparameters you found?

For the lower band, we searched all integers between 0 and 32. This is because we think that a lower band filter that is larger than 13 could lead to loss of motor intention data. We though

that this boundary would be lower, but because everyone's brain functions differently, we wanted to experiment with a range that was a little outside of what we thought would be ideal.

For the upper band, we searched all integers between 18 and 35. As mentioned earlier, we are interested in motor intention data which we expect to be between 4 and 30 Hz. To leave some room for unexpected behavior in the data, we searched a bit above 30 Hz, resulting in an upper band filter of 35 Hz.

For the number of CSP components, we searched the multiples of 2 in the range between 2 and 10. As discussed earlier, multiples of two are logical as we are trying to differentiate two classes, and we want to search a larger space than we expect to see as the participant's data could differ from our expectations.

We found that the ideal lower band was around 8Hz, the ideal upper band did not matter much for the overall performance, but the highest performance was found around 19Hz, which seems strange as a large part of the beta channel is then cut off. The ideal number of CSP was found to be 6.

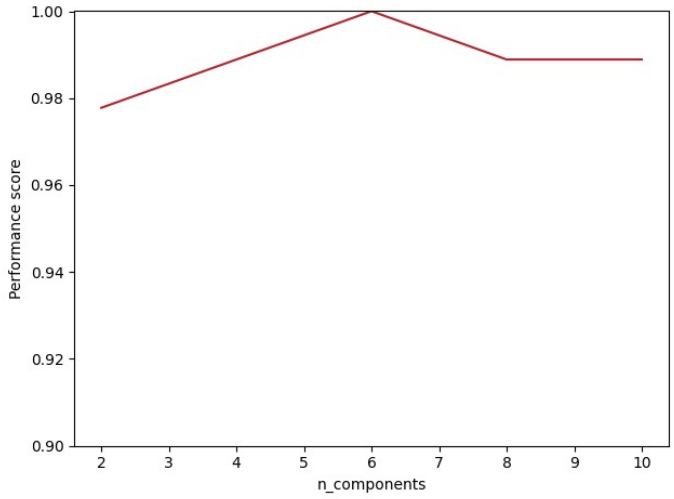

Figure 6: The performance score with a lower band of 4 Hz and an upper band of 30 Hz. The maximum score can be found at $n\_components = 6$.

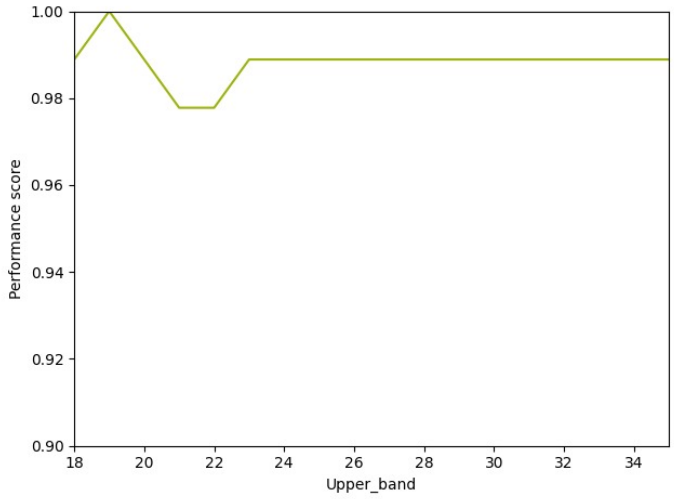

Figure 7: The performance score with a lower band of 4 Hz, and 4 CSP components. The maximum score can be found at upper_band=19

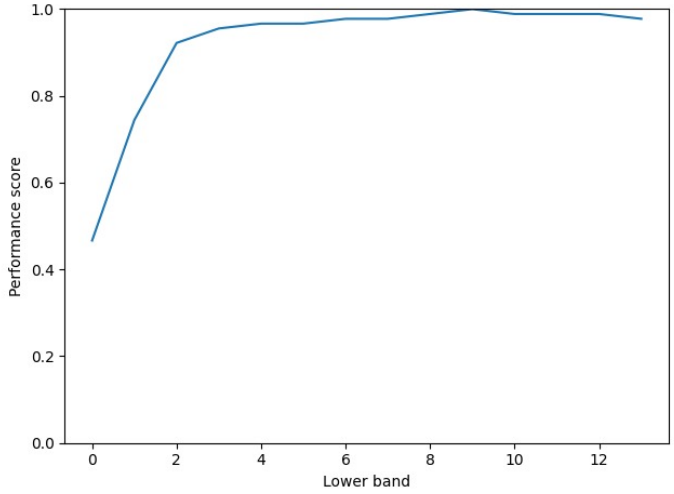

Figure 8: The performance score with an upper band of 30Hz, and 4 CSP components. The maximum score can be found at lower_band = 9Hz

Part 2

Step 1 [1 point]

Make covariance matrices for each epoch. The EEG data X from Part 1 is given as a numpy array of trials x channels x samples. For each trial take the covariance ma-

trix between channels, using the samples as the observations. This should give you 90 covariance matrices of 64 × 64. Then, take the average covariance matrices for all samples where y = 0 (left hand) and where y = 1 (right hand). Use plt.matshow() to visualize them. What do you observe, are there any clear differences? Please include the plots in your submission. (hint: it may be that you do not find much difference)

Firstly we notice that the covariance matrices have symmetry along the diagonal axis. This is logical, as they compare different channels to each other. Thus the same channels get compared twice. This means that the diagonal (top-left to bottom-right) gives us the variance per channel. We can see a slightly brighter matrix for the left hand, so there are higher covariances between channels during motor intention tasks for the left. This means that these channels are often activated in a similar fashion, indicating a similar function or source.

We found that there is little difference between the left and right hands.

There are also black lines in the matrix, this is a covariance of zero. This means there is no correlating behavior between this channel and the others, probably indicating noise-like activation.

In the covariance matrices, we see that there is a higher covariance between channels that are near each other, especially between the channels between 45 and 64. These channels are near the occipital lobe of the brain. This might be an indication of muscle noise in the neck, which is near the occipital lobe.

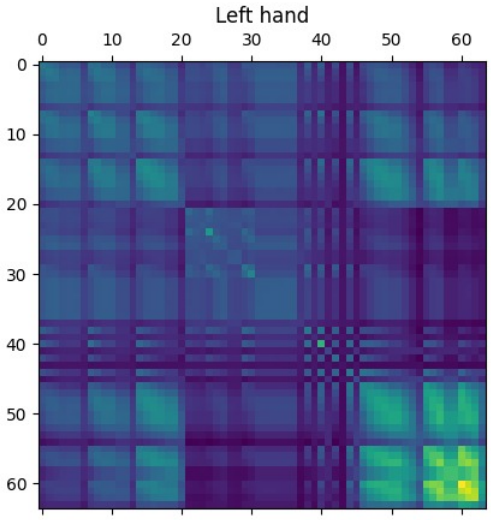

Figure 9: The covariance matrix of the left hand

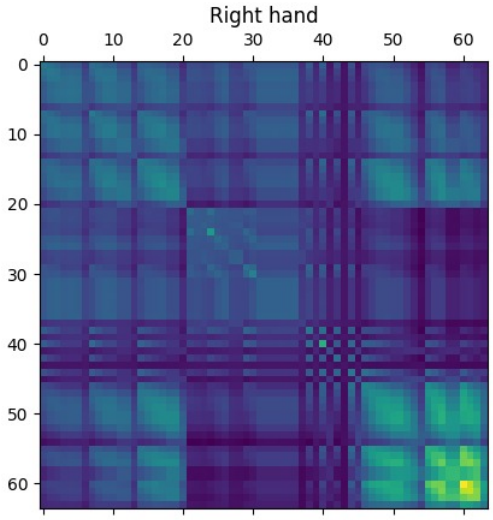

Figure 10: The covariance matrix of the right hand

Step 2 [2.5 points]

Split your covariance matrices and y into train and test sets using train test split(). Implement the K-Nearest Neighbours algorithm based on the covariance matrices. The distance should be determined with covariance distance(cov 1, cov 2, metric='riemann'). Then evaluate your Riemannian-KNN with ROC-AUC. What ROC-AUC did you get

and how does this compare against CSP-LDA? What value for k did you choose?

We arbitrarily chose k=5, as we expect this value to not be too small to underfit nor too large to overfit with a training set with n_samples=45. This resulted in a ROC-AUC score of 0.744 using Riemann distance

While it works as a classifier, the KNN (K=5, using Riemann distance) algorithm performed considerably worse than a tuned CSP-LDA classifier, which predicted around 20% better than the KNN classifier.

```
for method in ["riem", "euc"]:
    predictions = []
    for i, sample_x in enumerate(test_X):
        prediction = predict(sample_x, train_X, train_y, method, k=5)
        predictions.append(prediction)
    print(f'ROC-AUC using {method}: {roc_auc_score(test_y, predictions)}')

def predict(prediction_point, train_points, train_labels, metric, k):
    distances = []
    for train_point in train_points:
        # Define a function, bit redundant but alright
        if metric=="euc":
            dist = euc(prediction_point, train_point)
        elif metric=="riem":
            dist = riem(prediction_point, train_point)
        else:
            raise("AssertionError")
        distances.append(dist)

    dist_array = np.asarray(distances)
    closest_point_indices = np.argpartition(dist_array, k)[:k]
    left_or_right, counts = np.unique(train_labels[closest_point_indices],
                                      return_counts=True)

    most_frequent_class_index = np.argmax(counts)
    return left_or_right[most_frequent_class_index]
```

Step 3 [1 point]

Perform hyperparameter optimization for the number of neighbors (k) to be considered by KNN. Which values did you consider and why? What is the optimal value for k? Please include a plot showing the ROC-AUC on the y-axis, and the value k on the on the x-axis.

The range we considered is between 1 and 30. We thought that values over 30 would cause the model to always predict the most prominent class, as we have a total of 45 samples. That is, the class that occurs most often would always be predicted. Lower than 1 is impossible and we went as low as 1 because it could be that the most similar (=smallest covariance distance) matrix would simply be of the same class. In that case, we assume a relatively clean-cut separation between the two classes. Because we think this is unlikely, we opted for K=5 in the initial guess.

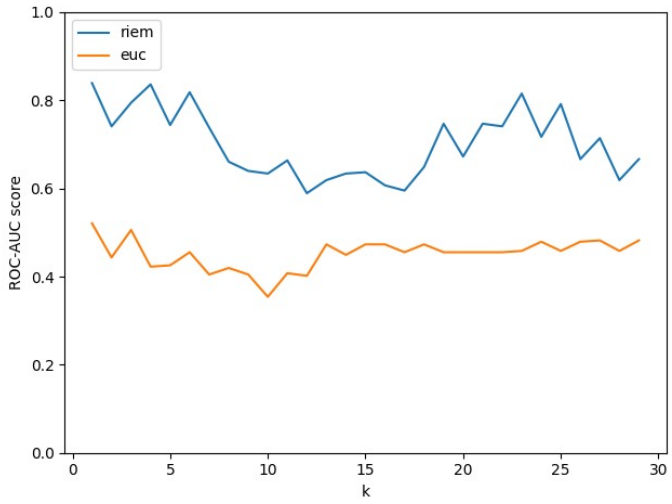

Figure 11: ROC-AUC scores for KNN using Riemann (riem) and Euclidean distance (euc), with the K-value along the x-axis.

Figure 11 shows that the optimal value can be found at K=4 for the Riemann distance, resulting in a ROC-AUC score of 0.842.

Step 4 [0.5 points]

Replace the Riemannian distance with (normal) Euclidean distance. How does this affect your KNN model's performance?

Using Euclidean distance, we get a ROC-AUC score of around 0.5. From Figure 11 we can see that the optimal performance is found at K=1, resulting in a ROC-AUC score of 0.521.

The fact that the algorithm scored so low (across all K) with the Euclidean distance surprised us, as it predicts more wrong than right, which is odd in a classification task with 2 classes.

As expected, the Riemann distance scored better than Euclidean, as this metric is based on positive-definite manifolds, which the covariance matrices are.

Step 5 [1 point]

Select a different subject at the start of the notebook. Again, optimize the bandpass filter for CSP-LDA, and the number of CSP filters. Are you finding the same optimal values?

For this question, we picked subject 20. The untuned CSP-LDA classifier had an ROC-AUC score of 0.338.

Using our found optimal hyperparameters (lower band of 9 Hz, upper band of 18 Hz, 6 CSP components), it had an ROC-AUC score of 0.488.

Using the same hyperparameter search ranges as in the previous question, we found the optimal parameters as follows:

	Optimal value	Score
Lower Bound	12	0.64
Upper Bound	34	0.63
N_components	2	0.63

Table 1: Optimal values and their respective ROC-AUC scores, for the lower bound, upper bound, and number of components

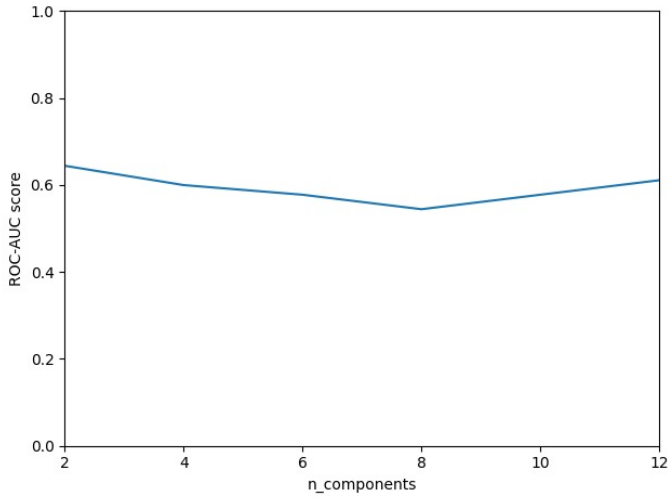

Figure 12: The ROC-AUC score when varying the number of CSP components

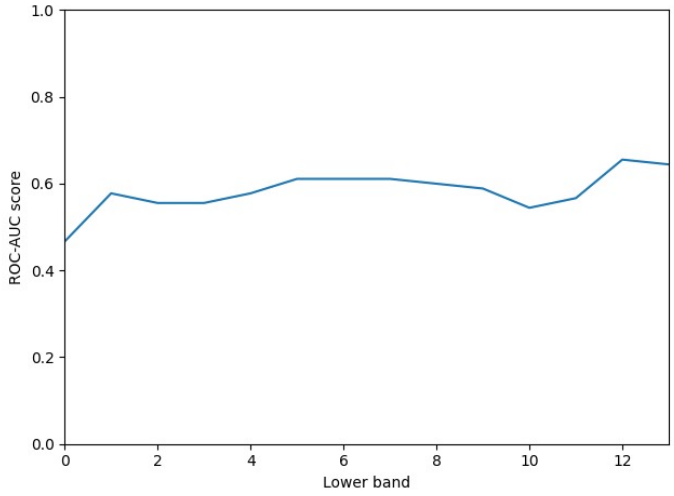

Figure 13: The ROC-AUC score when varying the lower band

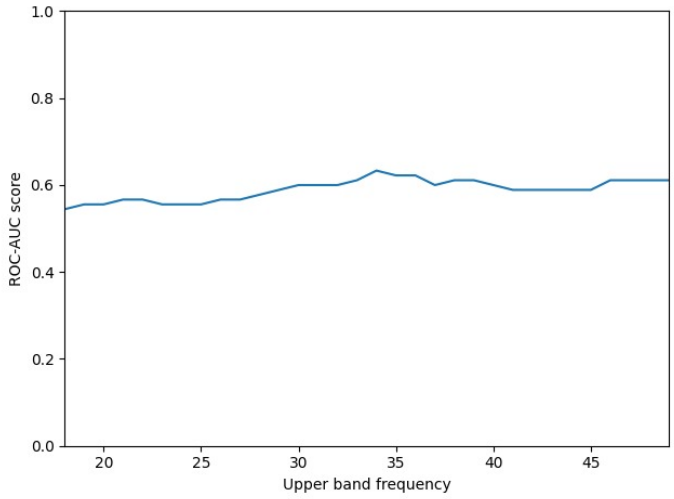

Figure 14: The ROC-AUC score when varying the upper band

Using these findings, we set the lower band to 12 Hz, the upper band to 34 Hz, and the number of CSP components to 2. This resulted in a model that achieved a ROC-AUC score of 0.589.

The optimal hyperparameters are quite different from our previous findings, as are these results.

Again apply KNN and find the optimal value for k. Are the optimal hyperparameters consistent for different subjects?

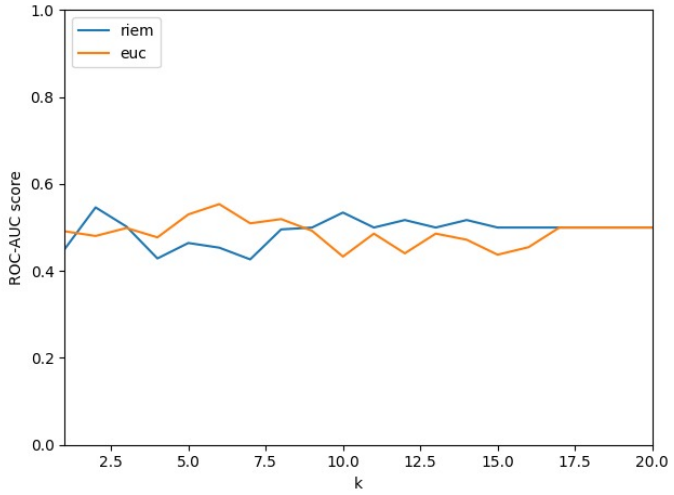

Figure 15: ROC-AUC scores for KNN using Riemann (riem) and Euclidean distance (euc), with the K-value along the x-axis.

From Figure 15, we can see that the optimal K for Riemman distance is K=2, resulting in an

ROC-AUC score of 0.551. For Euclidean distance, the optimal K=6, resulting in an ROC-AUC score of 0.558.

Surprisingly, Euclidean distance now outperforms Riemann distance. However, it must be said that we only ran the experiment once with these hyperparameters and the dataset is quite small so these results are not conclusive.

What does that say about the optimal hyperparameters we found in this assignment? From the previous results, we can see that the hyperparameters we found in the first part of this assignment are strictly related to the person we found them for. This tells us that each person will have different ranges of EEG data frequencies, as well as different patterns in their EEG data. Also, models need to be tuned individually for each person, and some people have stronger patterns in their EEG data, making them more suitable candidates for classifying models.