Brain-Computer Interface Movement Decoding

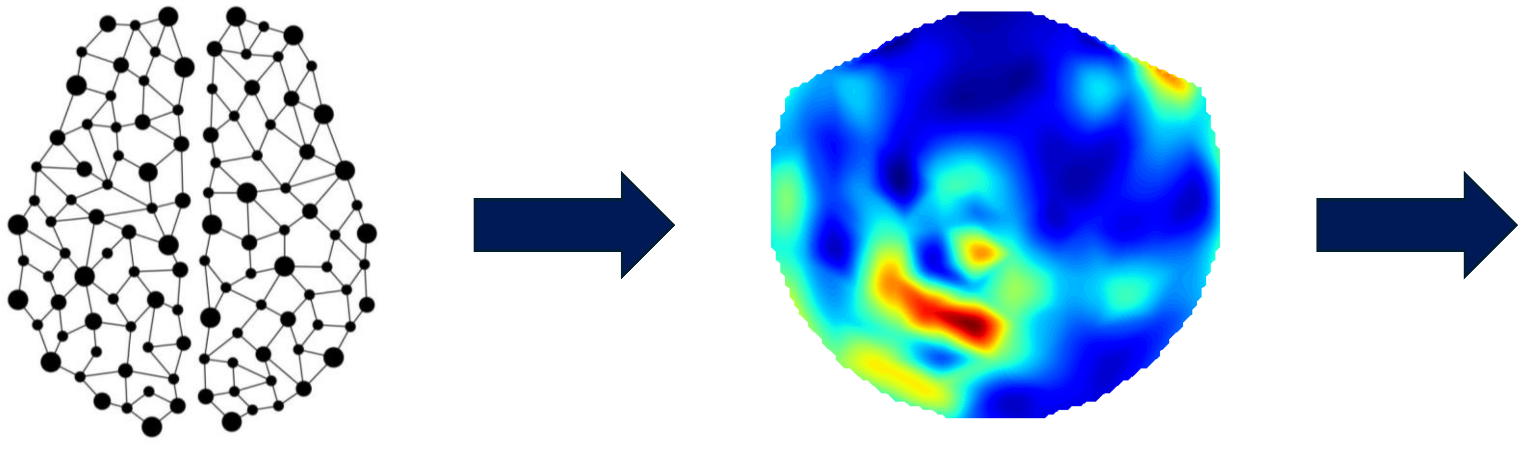
ECE 580 Mini Project 2

Bryan Boyd

Duke University

Problem Description - Overview

- The goal of this project is to create an algorithm using a support vector machine (SVM) with Two-Level cross-validation which can determine whether someone using a brain-computer interface (BCI) intends to select a "left" or "right" movement.
- BCIs receive, process, and convert brain impulses into commands that are sent to output devices to perform desired tasks.[1] They are often used by individuals who lack the neuromuscular control to interact with their environment on their own, and can thus be benefitted by artificial control, still using their neurons. This is particularly useful for those with lack of neuromuscular control of their arms and/or legs. This is an important problem to solve, as the reintroduction of independent interactions with the environment for disabled individuals can be potentially life changing. Another reason why this is an interesting problem is because this presents us with the task of applying an SVM to classify high-dimensional real-world data.



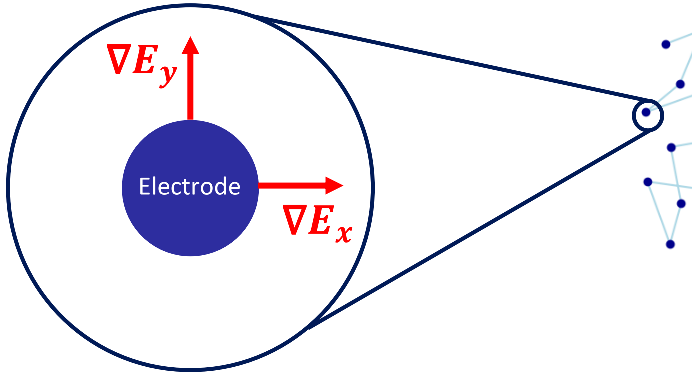
Duke

https://www.vecteezy.com/vector-art/3076913-human-brain-from-nodes-and-connections

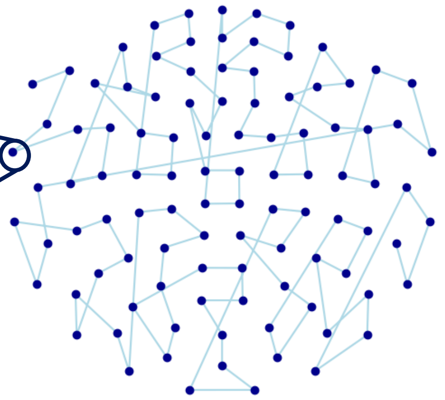
https://www.researchgate.net/figure/Figure-Bionic-Arms-Neurorobotic-Sensory-Motor-Fusion-Restores-Natural-Behaviors_fig6_360317362

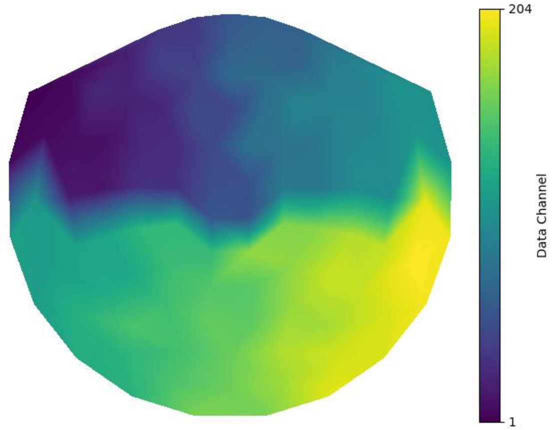
Problem Description – Project Data

- For this project, the data comes from electroencephalograms (EEGs) taken from test subjects. An EEG is a test that detects electrical activity of a subject's brain via the usage of electrodes adhered to the subject's scalp.[2] For these specific test subjects, there were 102 electrodes placed on the scalp, and each electrode provides two data channels of information – one related to the gradient of the electric field in the x-direction and the other related to the gradient of the electric field in the y-direction. This means for each test, there are 204 unique features.
- As for the testing, there were two different types of trials done – overt and imagined movements. Overt movement trials involved the participant moving either solely their left arm or right arm. Imagined movements however involved the participant imagining that they were moving either their left or right arm. For each combination of movement type (overt/imagined) and arm direction (left/right), there were 120 trials taken. It is important to note that we do not know which dataset 1 or 2 was left or right, so for the rest of this project we refer to the movements as either Movement 1 or Movement 2.



Each electrode has two data components


Scatter plot of 102 electrodes on brain for all trials

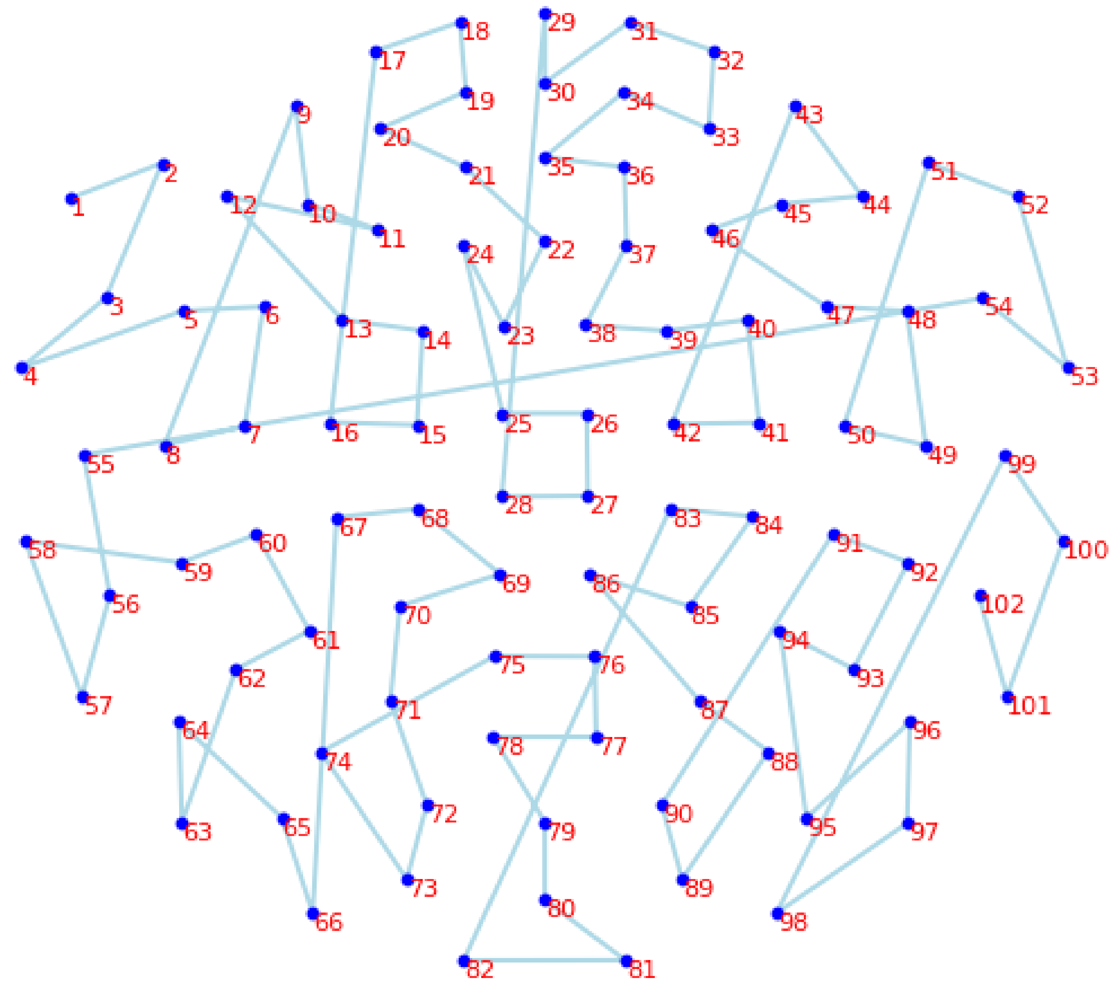


Visualization of the 204 data elements on the brain

Duke

Problem Description – Project Data

- In the simulations section of the report, I discuss specific electrode numbers, so here I provide the numbering of the electrodes at each location, to provide some context to these results.



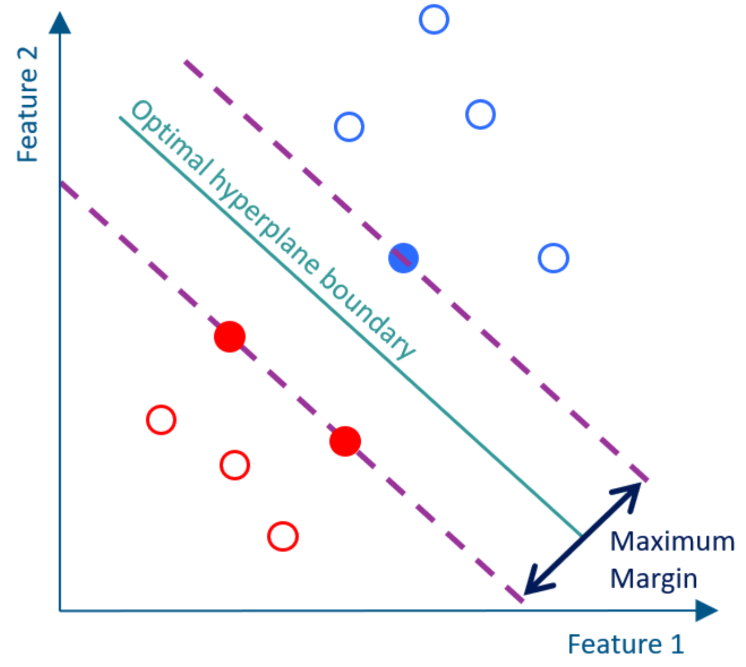
Duke

Problem Description – Why SVM? And Other Application Areas

- As stated previously, we have 204 features, and at most 240 trials amongst our entire data set, meaning we only have a few more trials than we do features, which can be difficult for accurate classification. SVMs however are known for their capability to handle high-dimensional data, since the classifier is built upon generating a hyperplane that maximizes separability.[3] This essentially is a built-in form of regularization, and through linear kernel SVM we can also implement a regularization parameter that is a penalty for misclassification and will be discussed more in later slides. Additionally, SVMs allow for different kernels for different models, allowing for more diversity in the potential classification boundaries.
- Within the medical field, an example of an SVM used for classification of high-dimensional data comes in the form of classifying different types of cancer, where thousands of genes represent different features.[4] Additionally, SVMs have been used for classifying Alzheimer's patients from those with mild cognitive impairment, using data from MRIs as features.[5]
- Even outside of the medical field there are many other potential application areas for this sort of SVM classification, such as within civil engineering, where it can be used to use high-dimensional features to classify damaged vs nondamaged components within structures.[6]

Mathematical Formulation - SVM Basics[7]

- A support vector machine (SVM) is a machine learning classifier which takes multidimensional data in the feature space and creates an optimal hyperplane boundary between the two classes which seeks to maximize the margin between the optimal hyperplane and the data points.
- Support vector machines can have multiple different types of kernels, but for this project's binary classification we use a linear kernel, and thus the optimal hyperplane is linear.
- SVMs determine the maximum margin, as well as the optimal hyperplane with the usage of only a few of the data points, known as support vectors, which are the colored in points on the plot shown to the right.
- As can be seen on the plot to the right, in the case of this linearly separable data set, only 3 points are support vectors and thus required to make the SVM, making this classifier resistant to the impact of outliers, and efficient for high dimensional data.
- For ease of calculation, SVMs typically assign class values of -1 and 1 rather than 0 and 1.



Duke

Mathematical Formulation - SVM Margin Maximization[7]

- The equation for the decision function for a general SVM classifier regardless of kernel type is the following linear model:

  $$\mathrm{y}(\mathbf{x}) = \mathbf{w}^{\mathbf{T}}\, \boldsymbol{\varphi}(\mathbf{x}) + \mathrm{b}$$

  where $\boldsymbol{w}$ represents the weight vector, $\boldsymbol{\varphi}(\boldsymbol{x})$ represents a fixed feature-space transformation, and b is the bias parameter. For a linear kernel $\boldsymbol{\varphi}(\boldsymbol{x}) = \boldsymbol{x}$, and since that is the primary kernel used for this project, further formulation will use the formula $y(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b$.
- Essentially, when we set $y(\boldsymbol{x}) = 0$, we get the equation of our hyperplane, if $y(\boldsymbol{x}) > 0$, the data point lays on one side of the hyperplane (the class with value 1), and if $y(\boldsymbol{x}) < 0$, the data point lays on the other side of the hyperplane (the class with value -1).
- We can represent the unsigned Euclidean distance of a data point $\boldsymbol{x_n}$ from the hyperplane with the following expression:

  $$\frac{t_n\,(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x_n} + b)}{\|\boldsymbol{w}\|}$$

  where $t_n \in \{+1, -1\}$ depending on class. If $\boldsymbol{x_1}$ is in Class 1, t_1 = +1. $\boldsymbol{x_1}$ is in Class -1, t_1 = -1.

- To find the maximum margin we want to maximize the smallest distance from any data point to the hyperplane. This turns into the following optimization problem:

  $$\underset{\boldsymbol{w},b}{\operatorname{argmax}}\left\{\frac{1}{\|\boldsymbol{w}\|}\min_{n}[t_n(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x_n} + b)]\right\}$$

Duke

Mathematical Formulation - SVM Optimization[7]

- The equation shown on the previous slide in its current form cannot be readily solved, so we look to rewrite it as an equivalent optimization problem shown below:

$$\underset{\boldsymbol{w},b}{\operatorname{argmax}}\left\{\frac{1}{||\boldsymbol{w}||}\right\}\ \textit{subject to}\ t_n(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{\boldsymbol{n}}+b)\ \geq 1\ \forall n$$

  which enforces that the unsigned distance from each data point to the optimal boundary is ≥ 1.

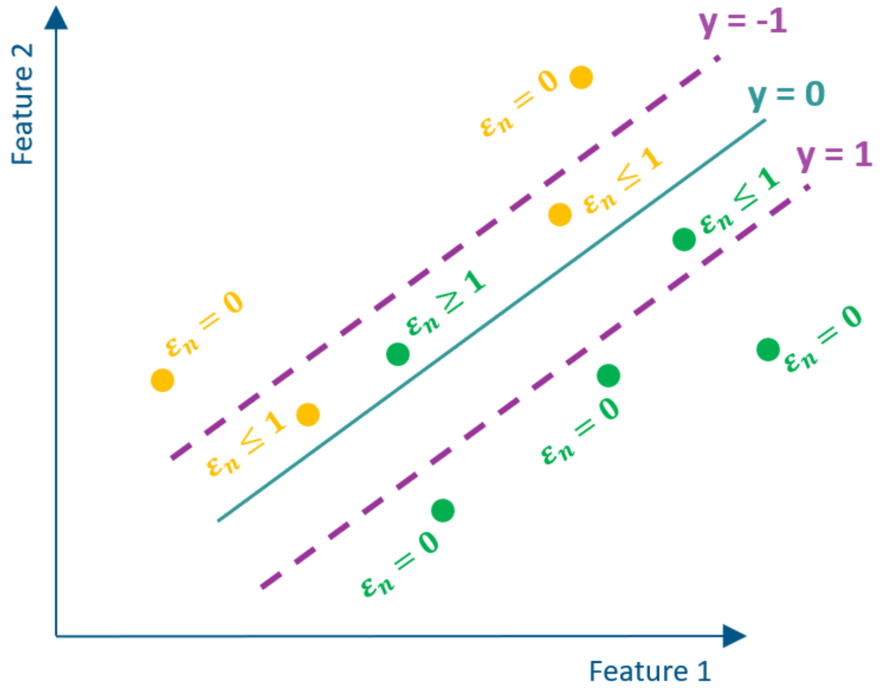
- Since the maximum of $\frac{1}{||\boldsymbol{w}||}$ is equivalent to the minimum of $\frac{1}{2}||\boldsymbol{w}||^2$, we can rewrite this optimization problem as the following quadratic programming problem:

$$\underset{\boldsymbol{w},b}{\operatorname{arg}min}\left\{\frac{1}{2}||\boldsymbol{w}||^2\right\}\ \textit{subject to}\ t_n(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{\boldsymbol{n}}+b)\ \geq 1\ \forall n$$

  A quadratic programming problem is one in which the function being minimized is quadratic and subject to a set of linear inequality constraints, as we see with the problem above.

Mathematical Formulation - SVM Overlapping Data[7]

- The optimization problem on the previous slide is contingent upon the data being linearly separable. Since this is not always the case, to make the SVM more robust we introduce a slack variable (hinge loss) as a penalty for misclassification ε_n which is equal to 0 if the data point is classified correctly and outside of the margin, and equal to $|t_n - y(x_n)|$ if the data point is correctly classified but within the margin or incorrectly classified. Specifically, if the data point is correctly classified but inside the margin, $0 \le \varepsilon_n \le 1$, and if the data point is incorrectly classified $\varepsilon_n \ge 1$.
- The new goal of the optimization problem adds to the previous goal of maximizing the margin, but with this additional penalty for data points that are classified on the wrong side of the margin.



Duke

Mathematical Formulation - SVM Lagrangian Function[7]

- Thus, the new optimization problem that we are looking to solve becomes the following:

$$\underset{\boldsymbol{w},b}{\operatorname{argmin}} \left\{ \frac{1}{2} \|\boldsymbol{w}\|_{\boldsymbol{2}}{}^{2} + C \sum_{n=1}^{N} \varepsilon_n \right\} \; such\ that\ t_n\left(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x}_{\boldsymbol{n}} + b\right) - 1 + \varepsilon_n \geq 0\ and\ \varepsilon_n \geq 0$$

- The constant C represents the regularization parameter which handles the balance between the slack variable penalty and the margin.
- To solve the optimization problem introduced on the previous slide we can use a Lagrangian function defined by:

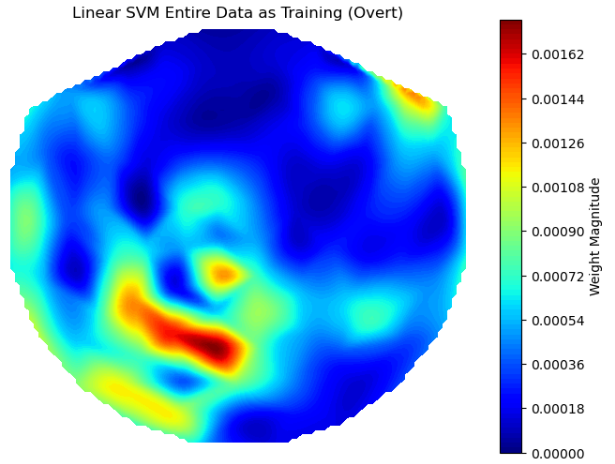
$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{a}, \mu) = \frac{1}{2} \|\boldsymbol{w}\|_{\boldsymbol{2}}{}^{2} - \sum_{n=1}^{N} a_n \{ t_n\left(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x}_{\boldsymbol{n}} + b\right) - 1 + \varepsilon_n \} - \sum_{n=1}^{N} \mu_n \varepsilon_n + C \sum_{n=1}^{N} \varepsilon_n$$

  where $a_n \geq 0$ and $\mu_n \geq 0$ are Lagrange multipliers.
- From here, using partial derivatives and KKT conditions, the dual formulation is solved for, which allows the enabling of the kernel trick which allows for the computation of the SVM. More details on this can be found in [7].

Duke

Implementation - SVM

- For this project, we were given 4 different datasets: feaSubEImg_1.csv, feaSubEImg_2.csv, feaSubEOvert_1.csv, and feaSubEOvert_2.csv. The two datasets ending in 1 one represent one direction of movement (either left or right), and the two datasets ending in 2 represent the other. As discussed previously Img would represent imagined movements and Overt represents overt movements.
- Each of the datasets contains 204 rows corresponding to the different channels, and 120 columns corresponding to the different trials. We transpose this data and then add a column to the beginning indicating whether this trial was for a Class 1 movement or a Class 2 movement. Now each row represents a trial for a specific class movement, and the EEG values for the 204 channels for that trial, thus we have 204 features.
- The specifics of the exact testing/training splits will be discussed within the next few slides, but data is often reported as showing a spatial plot of the weight magnitudes at the electrode locations. The weight magnitudes were found by taking $\sqrt{{w_x}^2 + {w_y}^2}$ for each electrode (aka taking the L2 norm of the 2 channel weights for a given electrode).
- The plots to the right show a basic visualization of the Linear SVM outputs for overt movement, using the entire data set for training, as well as a regularization parameter of 1. This is just a sanity check before beginning 2-level cross-validation. The SVM classifiers for this project are implemented using scikit-learn's SVC class with specified kernels and regularization parameters, since this method inherently uses a hinge loss function as well as L2-norm penalty.



Duke

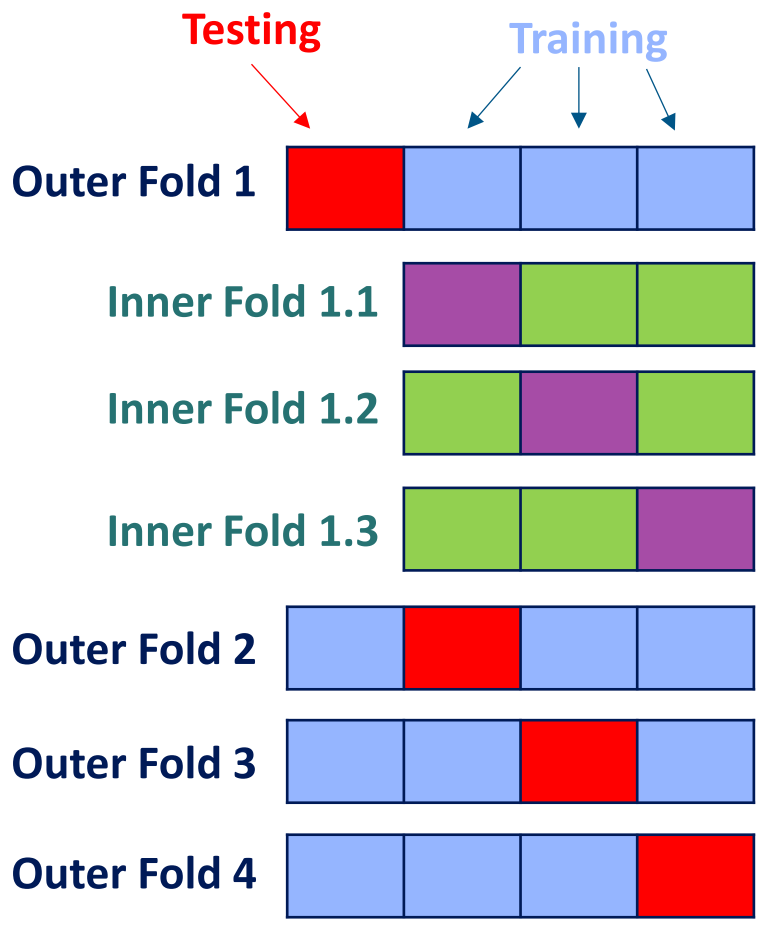
Simulations - Overview

The next section goes through most of the simulations done throughout this project, which for both movement types includes:

- Determination the optimal regularization parameter **C** per fold using two-fold cross-validataion.
- Receiver Operating Characteristics (ROC) plots for each first level cross-validation fold as well as the total cross-validated ROC.
- Accuracies for each first level cross-validation fold using the optimal regularization parameter determined from the second-level cross-validation, as well as the total cross-validated accuracy.
- Visualizations of the magnitude of channel weights on brain surface, as well as the signed weights for every channel for the first fold of the first level cross-validation.
- Cross-training simulations (training on imagined movements testing on overt movements, and training on overt movements and testing on imagined movements).
- Accuracies of training on different kernels other than linear (radial basis function, polynomial, sigmoid).

Simulations – Two-Level Cross-Validation

- Typical single level K-fold cross-validation involves splitting up a dataset into K different sections and taking turns using section 1 through K as the testing data, and all the other K-1 sections are used as training data. The performance of each of the different testing/training splits can be taken to give a better understanding of your data without needing to take more data. However, this may lead to some bias since the same folds are being used for both choosing the best hyperparameter and estimating the model's performance.
- Two-level cross-validation solves the previously stated problem of using the same folds for both optimizing your parameter and measuring performance, and splits it into two loops, and outer and inner loop. Essentially, you first split your data into K folds just like previously, except now, the 1 fold that is set aside as testing is completely removed for the moment. The other K-1 folds are used for N-Fold (N = K – 1) cross-validation to determine the best hyperparameter value. Then after the inner loop of N-fold cross-validation is completed, that optimal hyperparameter is tested using the one outer test fold that was never used in the training. A general example is shown to the right.



Each inner fold i.j gets its own optimal hyperparameter, and the best across the three inner folds gets tested on outer fold i.

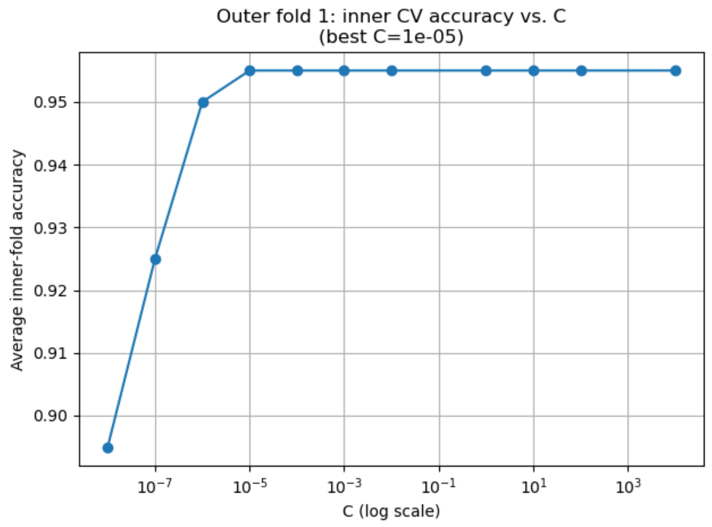
Duke

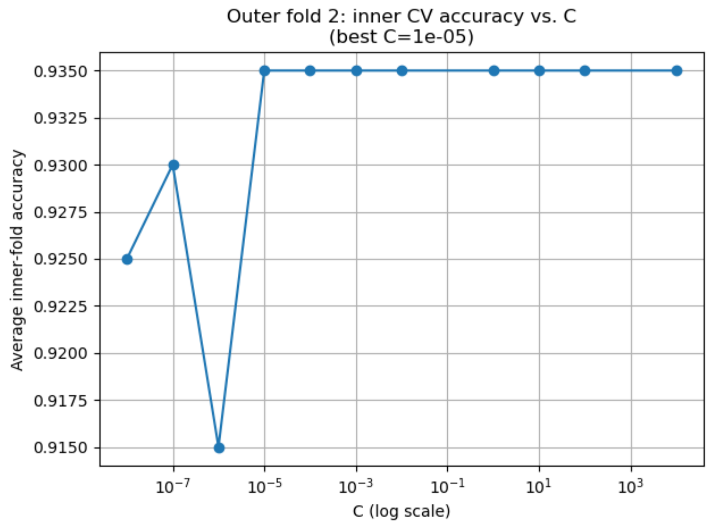
Simulations – Two-Level CV Applied

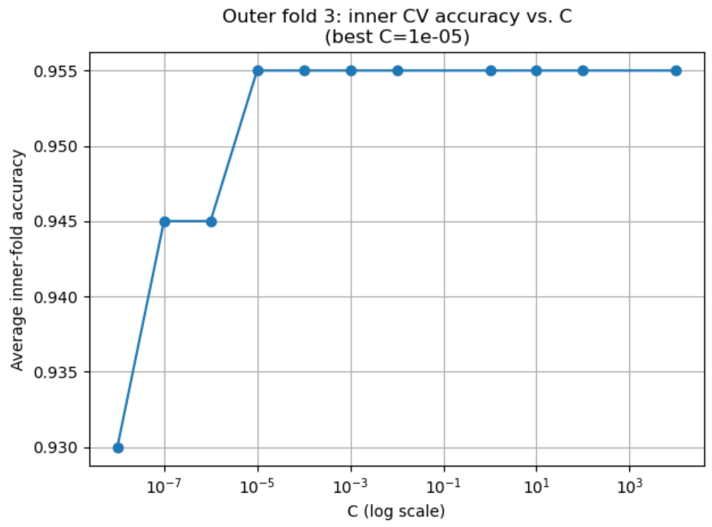
- Specifically for this project, two-level cross-validation is used when testing and training on the same movement type (overt or imagined). Each data set consists of 240 trials (consisting of 120 movement 1 trials and 120 movement 2 trials).
- For the first level the data is divided into six stratified folds, which results in 40 trials in each fold, 20 per movement class. This means that training data consists of 200 trials and testing consists of 40 trials. The splitting
- For the second level 5-fold cross validation is done to determine the optimal regularization parameter **C** for that first level fold. This means for each fold in the second level, there are 160 trials being used for training and 40 trials being used for testing. The regularization values being tested include 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1, 1e1, 1e2, and 1e4. For each inner fold, all these regularization parameters were tested, and the average accuracy across all the 5 second level folds were determined. The regularization parameter with the highest average accuracy amongst all the 5 second level folds for a given first level fold was chosen as the regularization parameter for that first level fold. The plots of average accuracy amongst the second level cross-validation for each first level fold are shown on the next two slides.
- For cross-validation, scikit-learn's StratifiedKFold and PredefinedSplit were utilized.
- As discussed above, the choice of hyperparameter is dependent on accuracy. What is referred to here as accuracy is the number of correctly classified data points divided by the total number of data points. This is determined using scikit-learn's score() function for SVC().

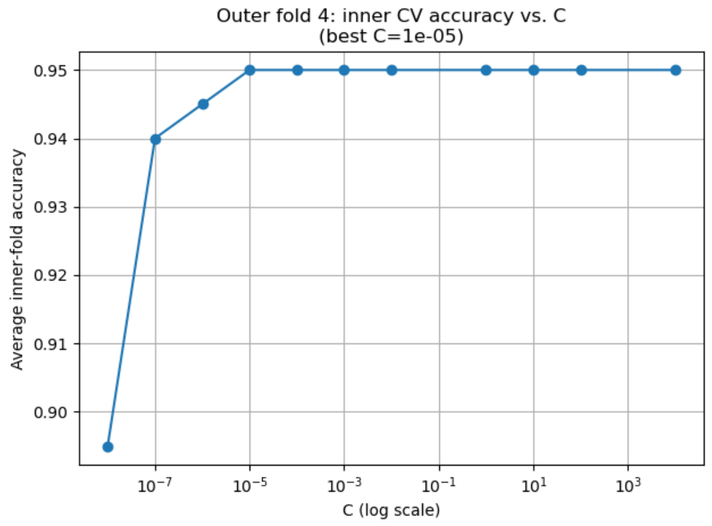
Note: Outer fold and first level fold are used interchangeably throughout this presentation. Inner fold and second level fold are also used interchangeably.

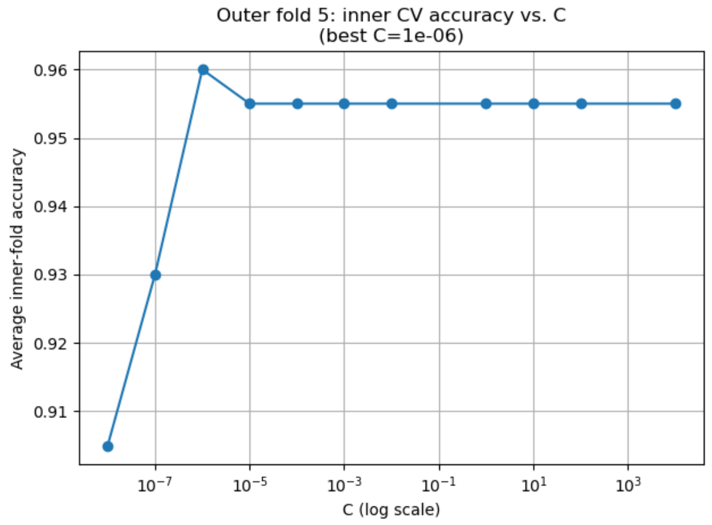
Simulations – **C** optimization – Overt Movement

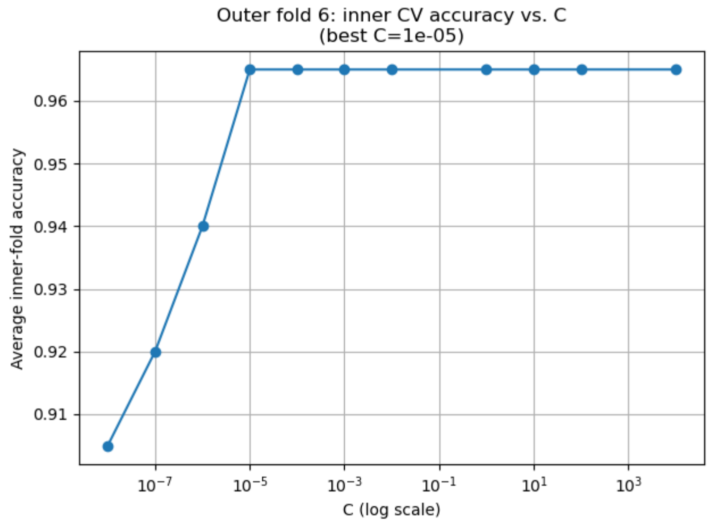






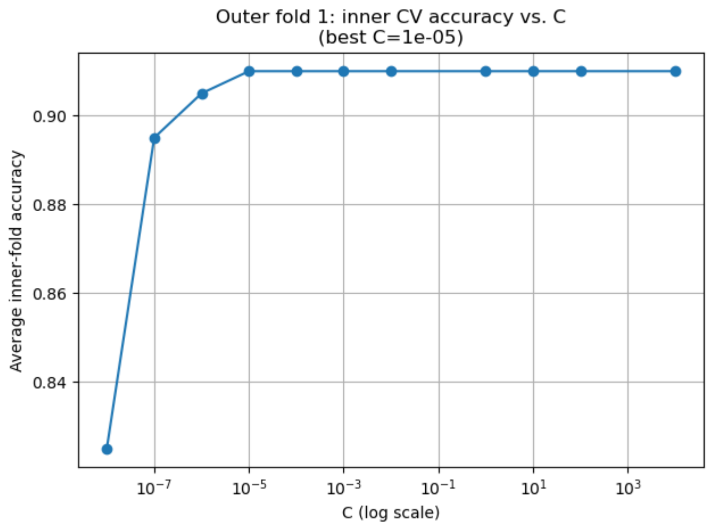


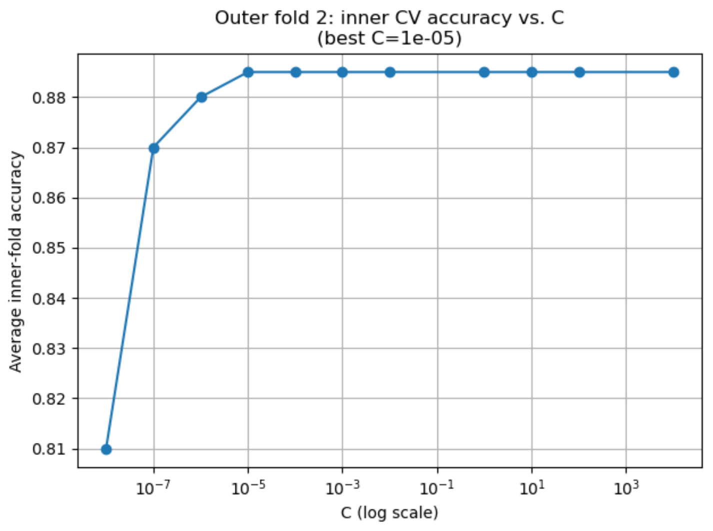


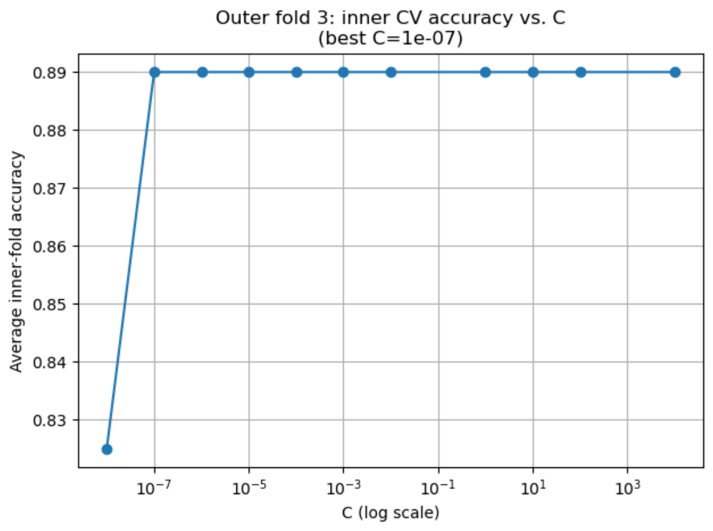


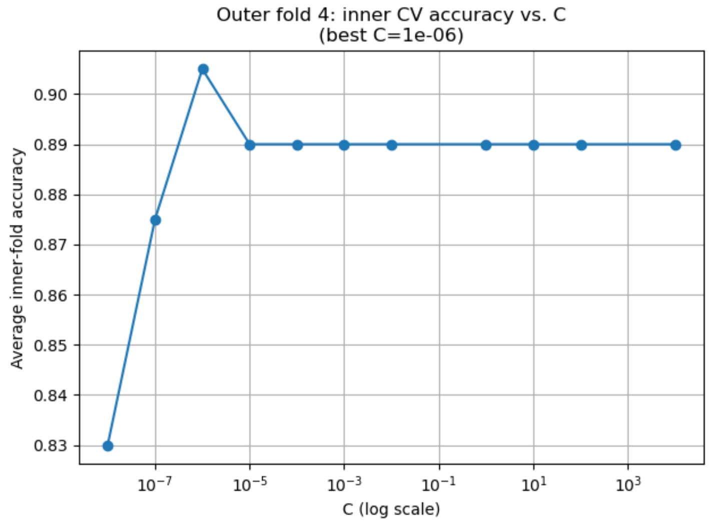
Duke

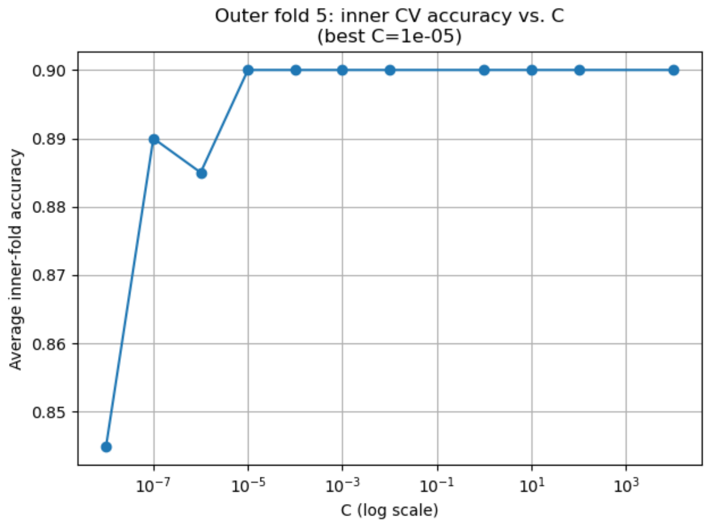
Simulations – **C** optimization – Imagined Movement

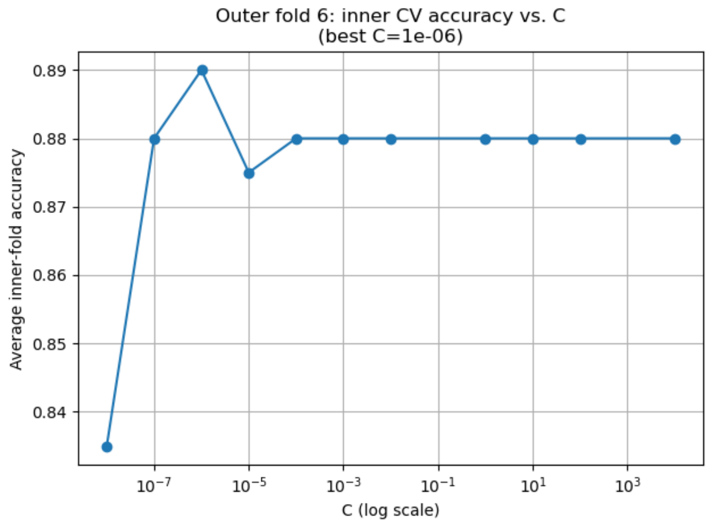












Duke

Simulations – First Level Cross-Validation Accuracy Results

- As shown on the previous two slides, for each first level fold, the regularization parameter **C** that maximized the accuracy for the average of the second level folds was utilized here for both movement types. Then this regularization parameter was used for the training of the linear SVM for that specific fold. The accuracy results are shown on the tables to the right.

- We see that the overt movement has a higher cross-validated accuracy compared to the imagined movement. This makes sense because we expect to see strong signals from the EEG when a person is actually doing a movement, compared to when they are simply imagining one. With more prominent signal strengths, we would likely find it easier to classify over many trials, thus we would expect a higher accuracy for overt movements compared to imagined movements.

- We see that the cross-validated accuracies are quite representative of the individual fold accuracies, with there not really being any outliers amongst the folds for either of the movement types.

- For cross-training purposes based on the results, we will use C = 1e-05 for training on overt movements and C = 1e-06 for training on imagined movements.

Overt Movement

First Level Fold #	C value	Accuracy
1	1e-05	0.95
2	1e-05	1
3	1e-05	0.925
4	1e-05	0.975
5	1e-06	0.95
6	1e-05	0.975

Cross-validated accuracy = 0.962

Imagined Movement

First Level Fold #	C value	Accuracy
1	1e-05	0.825
2	1e-05	0.975
3	1e-07	0.85
4	1e-06	0.9
5	1e-05	0.85
6	1e-06	0.925

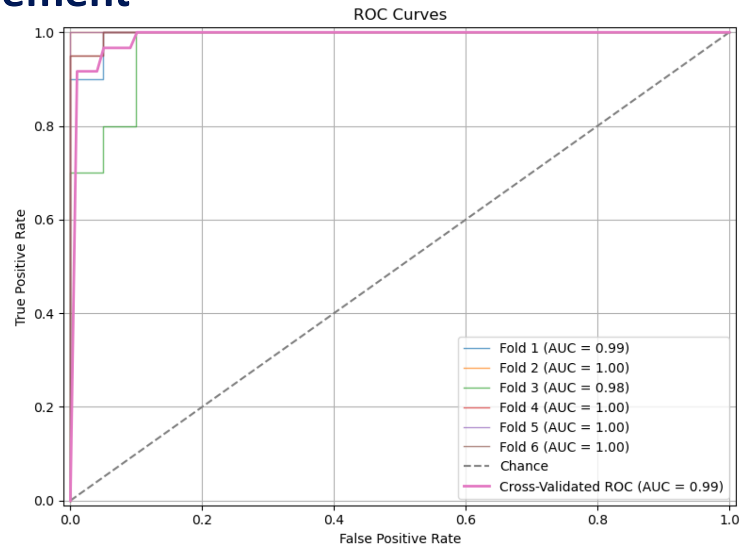
Cross-validated accuracy = 0.887

Duke

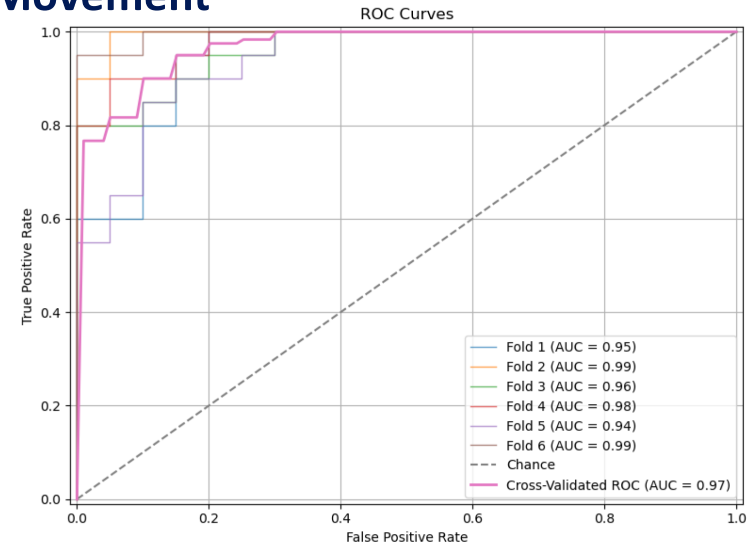
Simulations – First Level Cross-Validation ROC Results

- To the right we have the ROC curves for the 6 outer folds as well as the cross-validated ROC curve for each of the movement types.
- While both ROC curves produce areas under the curve (AUCs) very close to 1, indicating that our model is provides very good separability, we see that the cross-validated AUC for the overt movement is a bit higher than imagined movement, as it is also closer to the top-left corner, which is where we would expect a perfect model to lie.
- We notice that the individual fold ROCs are a bit more representative of the cross-validated ROC for the overt movement, as they tend to be more clustered (aside from one potential outlier) compared to the imagined movements which seem more spread out around the cross-validated ROC.
- The cross-validated ROC for overt movement being closer to the perfect model than the cross-validated ROC for imagined movement makes sense for similar reasoning to the accuracy results on the previous page. Since the overt movements produce stronger signals, and likely more consistent, it would make sense for the ROC to be closer to perfect compared to that of imagined movements, in which the data is weaker and has some more variability, and thus that is reflected in its ROC.

Overt Movement



Imagined Movement

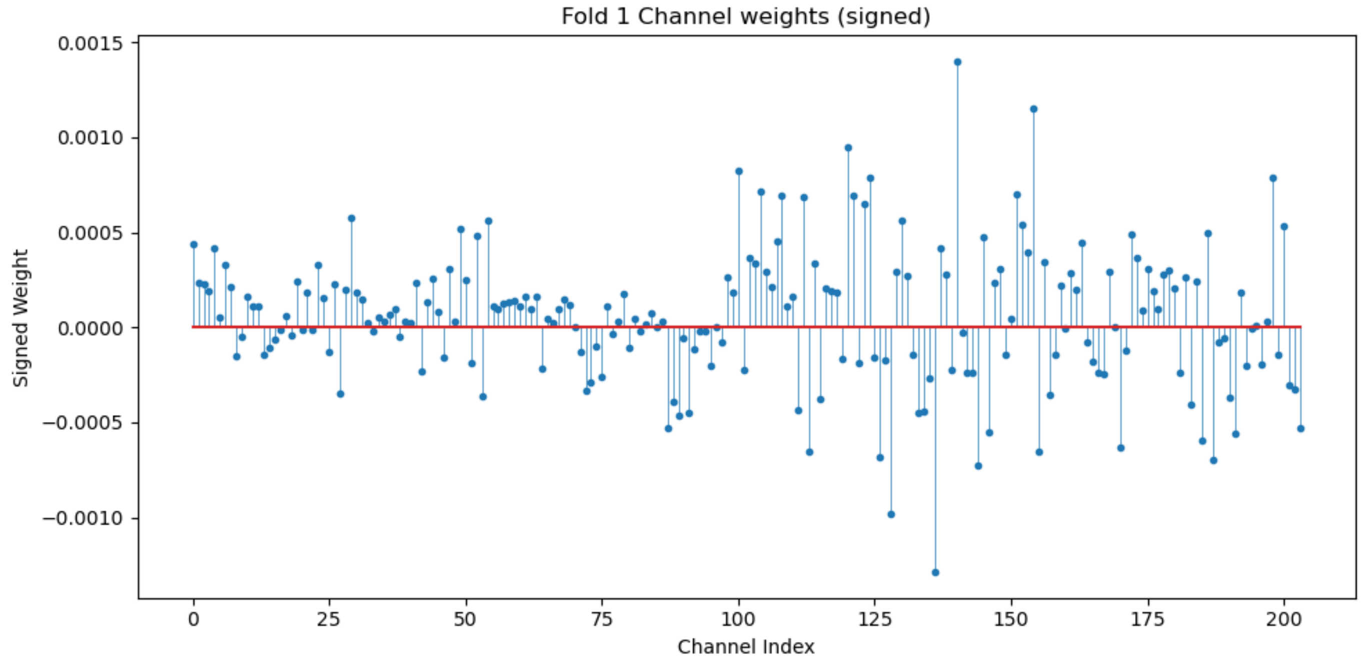


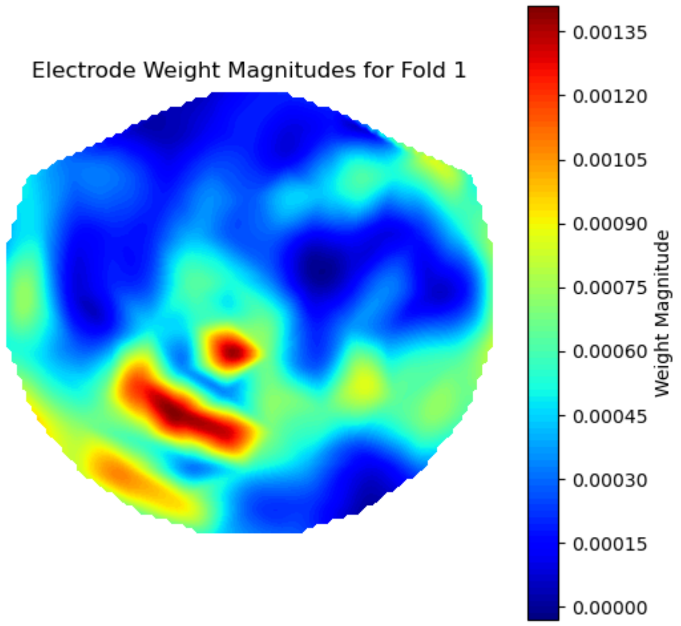
Duke

Simulations – First Level CV 1st Fold Results – Overt Movement

On this slide we see the results from training the first fold of the first level using the optimal regularization parameter determined the second level cross-validation for the overt movement set. We display the signed weight of the top six channels with the largest unsigned weights, a stem plot of all the channel weights, and a visualization of the electrode weight magnitudes for all 102 electrodes.

Top 6 channels by unsigned weight	
Channel #	Signed Weight
140	+0.0014
136	-0.0013
154	+0.0011
128	-0.0010
120	+0.0009
100	+0.0008



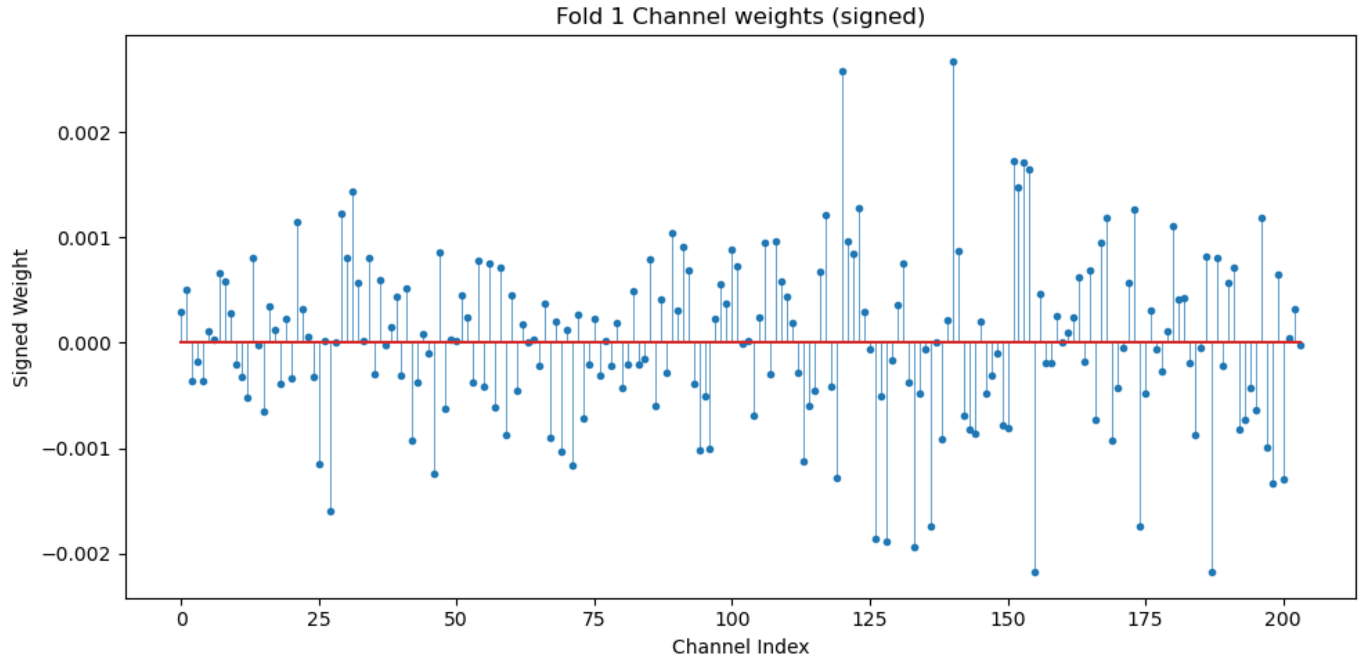


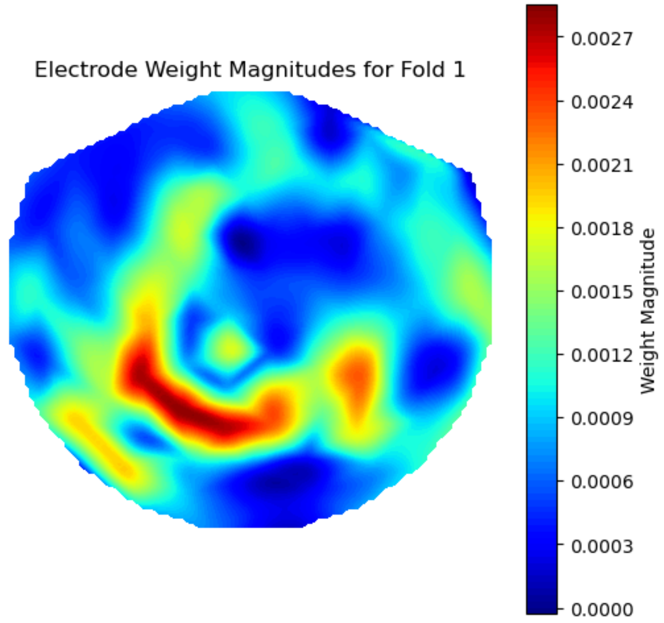
Duke

Simulations – First Level CV 1st Fold Results – Imagined Movement

On this slide we see the results from training the first fold of the first level using the optimal regularization parameter determined the second level cross-validation for the imagined movement set. We display the signed weight of the top six channels with the largest unsigned weights, a stem plot of all the channel weights, and a visualization of the electrode weight magnitudes for all 102 electrodes.

Top 6 channels by unsigned weight	
Channel #	Signed Weight
140	+0.0027
120	+0.0026
155	-0.0022
187	-0.0022
133	-0.0019
128	-0.0019





Duke

Simulations – First Level CV 1st Fold Results – Comparisons

- Starting with the stem plots for the two different movement types (overt vs. imagined), we see that for the imagined movement fold, there seems to be a lot more activated channels. For the overt movement fold, it seems that there are more channels with weights closer to zero.

- Looking at the visualization, we see that it seems like more areas of the brain are activated for the imagined movement compared to the overt moment. For the overt movement we see more regions of deep blue which correspond to lower weight magnitudes, whereas we see a lot more of the lighter blue regions in the imagined movement results, indicating some middle weight magnitude.

- Finally, looking at the channels with the largest unsigned weights, we see that there are some channels which are common amongst the top six of the two (140, 120, 128). Additionally, we see that the largest channel weights are larger for the imagined movement fold compared to the overt movement fold.

- Taking all of this into account, it seems like there are certainly certain electrode locations which are more significant in determining which direction of movement (1 or 2) a user is making for both imagined and overt movements. We also see that there are more electrodes that are significant for classifying the movement for imagined movements, since the weights are typically larger throughout, which implies that the imagined dataset is more complex, and more complicated to model.

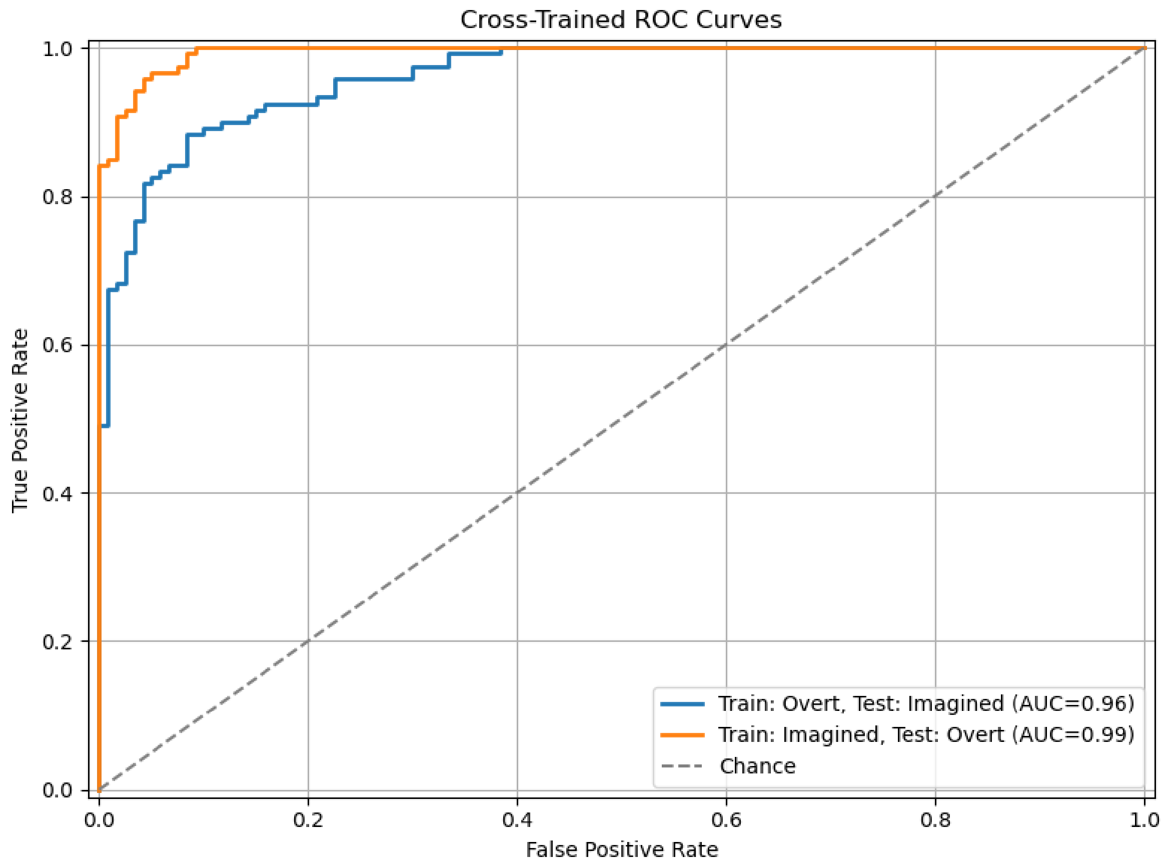
Duke

Simulations – Cross Training

- For this simulation, we used the optimal parameters after the two-level cross-validation for the overt training (1e-05) and the imagined training (1e-06) to train two separate models and tested those models on the opposite movement type (i.e. one was trained on the entire overt dataset and tested on the entire imagined dataset and vice versa).

- To the right we have the ROC curves and accuracies for both models. As we see, training on overt and testing on imagined gives us a worse result in terms of both accuracy and ROC curve compared to training on imagined and testing on overt.

- These results make sense, as the imagined dataset is seemingly more complex than the overt dataset, and while on the previous slides we saw that they share some of the same electrodes as significant ones, the imagined movement seems to assign higher weights to more channels than the overt movement. This may not be a big issue for testing the overt trained on the imagined but can cause issues for testing imagined trained on the overt if the overt doesn't give enough weight to the channels to make the appropriate decision for movement direction.



Accuracies:

Trained on overt, tested on imagined: 0.883

Trained on imagined, tested on overt: 0.950

Duke

Simulations – Comparing Different Kernels

- For the final simulation, the entire two-fold cross-validation was repeated to produce SVMs with different kernels rather than linear, to see if any other type of kernel would be more effective than linear. The other types of kernels tested were radial basis function (RBF), polynomial, and sigmoid. This was done for both overt and imagined movement datasets for all kernels. It is worth noting that only the regularization parameter was optimized, other parameters used in specific kernels were taken as the default values, which could skew results.

- As seen in the table below, the linear kernel produced the best results in terms of accuracy, while the other three models were still relatively accurate. Since we are already working in a large feature space (204 features), and we only have 120 trials for overt movements and 120 trials for imagined movements, it is possible that applying a transformation on these feature vectors that are already complex could be detrimental to the accuracy, meaning just a simple linear kernel could work best. It is also possible that the EEG data is just naturally more linearly separable compared to a more curved decision boundary.

Kernel	Overt Accuracy	Imagined Accuracy
Linear	0.962	0.887
RBF	0.925	0.875
Polynomial	0.904	0.821
Sigmoid	0.925	0.825

Duke

Conclusions - Overt vs. Imagined Summary

- Based on our results throughout the simulation section, it is evident that using SVMs to classify desired movement direction is possible and accurate for both overt and imagined movements, although it is more robust for overt movements. For the two-fold cross-validation done on for overt movement dataset we observe an accuracy 7.5% larger than the 88.7% accuracy achieved for the imagined movement dataset. It is likely that when you are physically moving your limb, you are more likely to be activating approximately the same neurons in your brain each time, compared to when you imagine moving your limb, you may be imagining it slightly differently each time, prompting a potentially weaker and more diverse neuron approach, making it harder to model. Past research confirms that it is typically harder to classify imagined movements compared to overt movements.[8]

- This idea that the imagined movement dataset may be more diverse and varying is also supported by the fact that for the first fold of the first level of the two-level cross-validation, there are more channels with significant weights in the stem plot compared the overt movement dataset, meaning that more channels are being recognized as significant for classification. This is potentially why we see it was difficult to test on imagined movements when training on either imagined or overt movements, and we get better results for testing overt movements when training on either imagined or overt movements, since seemingly less channels are significant for classifying overt movements.

Conclusions – Improvements Made and for the Future

- **Improvements made**
  - Widened the range of regularization parameters from the suggested minimum C = [0.01, 1, 100, 10000] to the more robust C = [1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1, 1e1, 1e2, 1e4]. This was certainly a beneficial improvement, since the optimal regularization parameter was a few orders of magnitude below the minimum of the original range, which improved our model.

  - Experimented with different kernel types (RBF, polynomial, and sigmoid). Even though none of these three kernels produced better results than the linear kernel, it confirms that we were using what was seemingly the best kernel for this problem.
- **Potential improvements for the future**
  - Experiment with using less features for classification that the entire 204 channels. For both ovet and imaginary movement datasets, it seems that the weights produced by the SVM for certain channels were very low, meaning they didn't have much impact on the model, and could likely be removed.

  - Experiment with other types of classification models such as relevance vector machines to see if we get better results.

Conclusions – Other Takeaways

- For both types of movements, using the visualizations and stem plots, we saw that certainly certain channels, and thus certain electrode locations were more significant for classification of movement direction. By linking this information with the contextual knowledge of which part of the brain is used for certain movement types, we could make even better insights into the model and determine if the significance of certain parameters is due to random error or potentially reduce the number of features in the model. This could also help provide insight into exactly which areas of the brain are driving the decision-making process for overt vs imaginary movements.

- The prescribed optimal regularization parameter for the overt movement dataset was C = 1e-05, whereas for the imagined movement dataset it was C = 1e-06. This difference in regularization parameter makes sense with our idea that the imagined dataset is generally more complex and noisier compared to the overt dataset. This difference means that we would want a stronger regularization (small C) for the imagined dataset to avoid overfitting and allow for a larger margin, compared to the overt dataset which we can be stricter with.

- As we previously discussed, using the imaginary movement for training and the overt movement for testing yields better results than using the overt movement for training and the imaginary movement for testing. This relates to the signal to noise ratio of the two datasets, as it seems the imaginary movement data set may have a lower signal to noise ratio (SNR) compared to the overt dataset. For future classification problem, I think it would be important to ensure that your training data has either equal or less signal to noise ratio compared to your expected dataset, since we saw the worst performing model in our case when we trained on the overt movement data (seemingly higher SNR) and tested on the imaginary movement data (seemingly lower SNR).

Collaborations

The only classmate whom I collaborated with was Jesen Tanadi. We often discussed various aspects of the project as we went along including the following:

- The difference in implementation between using scikit-learn’s LinearSVC() vs. SVC(kernel = ‘linear’)
- The exact meaning of two-fold implementation and how to go about determining what is considered to be the “optimal” regularization parameter, since there are multiple ways one could go about it.
- The specific expectations for reporting for this project.
- General conceptual questions about SVMs.

References - Readings

1. Shih, J. J., Krusienski, D. J., & Wolpaw, J. R. (2012). Brain-computer interfaces in medicine. Mayo Clinic Proceedings, 87(3), 268–279. https://doi.org/10.1016/j.mayocp.2011.12.008
2. https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/electroencephalogram-eeg
3. Ben-Hur, A., & Weston, J. (2009). A user's guide to support Vector Machines. Methods in Molecular Biology, 223–239. https://doi.org/10.1007/978-1-60327-241-4_13
4. Applications of support vector machine (SVM) learning in cancer genomics. (2018). Cancer Genomics &amp; Proteomics, 15(1). https://doi.org/10.21873/cgp.20063
5. Gerardin, E., Chételat, G., Chupin, M., Cuingnet, R., Desgranges, B., Kim, H.-S., Niethammer, M., Dubois, B., Lehéricy, S., Garnero, L., Eustache, F., & Colliot, O. (2009). Multidimensional classification of hippocampal shape features discriminates alzheimer's disease and mild cognitive impairment from normal aging. NeuroImage, 47(4), 1476–1486. https://doi.org/10.1016/j.neuroimage.2009.05.036
6. Pan, H., Azimi, M., Gui, G., Yan, F., & Lin, Z. (2017). Vibration-based support vector machine for Structural Health Monitoring. Lecture Notes in Civil Engineering, 167–178. https://doi.org/10.1007/978-3-319-67443-8_14

References - Readings

7. Bishop, C. M. (2006). Pattern recognition and machine learning. Springer New York.
8. Cho, J.-H., Jeong, J.-H., Kim, D.-J., & Lee, S.-W. (2020). A novel approach to classify natural grasp actions by estimating muscle activity patterns from EEG signals. CoRR, abs/2002.00556. Retrieved from https://arxiv.org/abs/2002.00556

Duke

References – Python Libraries

Python libraries used for the code include the following open-source libraries:

- Pandas – Organizing the data into data frames from csv files
  https://pandas.pydata.org/docs/
- Matplotlib – Used for plotting everything from visualization to scatter plots and ROCs
  https://matplotlib.org/stable/index.html
- Numpy – Used for manipulating data and mathematical operations
  https://numpy.org/doc/
- SciPy – Used for grid for visualization of weights on brain
  https://docs.scipy.org/doc/scipy/
- Scikit-learn – SVC – Used for computing the support vector machine for all kernel types
  https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- Scikit-learn – StratifiedKFold and PredefinedSplit – Used for two-level cross-validation
  https://scikit-learn.org/stable/api/sklearn.model_selection.html
- Scikit-learn – roc_curve and auc – Used to produce ROC curves
  https://scikit-learn.org/stable/api/sklearn.metrics.html

Duke