Machine Learning Approach to Identifying Neural Features That Predict Rodent Behavior

Final Report

Washington State University - Department of Psychology

Nunnerson Computing

Aidan Nunn, Charlie Nickerson
2/2/23

CptS 423 Software Design Project II
Spring 2023

TABLE OF CONTENTS

I. Introduction

Nunnerson Computing has been tasked with building machine learning models for our clients at the Washington State University Department of Psychology Behavioral Neuroscience Lab. Our main goal is to provide the lab with software that can generate a model using Lasso (Least Absolute Shrinkage and Selection Operator) which will predict when to collect data from Sprague-Dawley rodent test subjects. The lab's current method of data collection is time consuming, and by using machine learning we will streamline the data collection process and save time for the researchers. We will also be utilizing the Kamiak Cluster supercomputer to speed up data processing.

We have also been tasked with two intermediate goals. These are to build a model which predicts if a rodent test subject is dependent or non-dependent on alcohol and models which predict how much alcohol a rat has drank during an experiment session. These goals were assigned because the data required for the primary model was not ready when we began working with the lab.

I.1. Client and Mentor Information

Our client is the WSU Department of Psychology Behavioral Neuroscience Lab. Dr. Angela Henricks, the head of the lab, is our primary contact for the project. The model we build will be used by Dr. Henricks and her graduate student Kelly Hewitt, but could potentially be used by other teams if they take over the research. Our team mentor is Ananth Jillepalli, an assistant professor at WSU.

I.2. Client Background

The WSU Department of Psychology Behavioral Neuroscience Lab uses rodents to study the neurobiology of substance use and mental illness. One of their focuses is the difference in males and females regarding these topics. Much less research has been conducted on females compared to males, so they wish to identify what the differences are. Their eventual goal is to apply their current research to humans.

II. Team Members - Bios and Project Roles

Developer Bio: Charlie Nickerson is an undergraduate at Washington State University working towards a Bachelor's Degree in Computer Science. Mr. Nickerson is the co-developer of Nunnerson Computing and is responsible for programming, testing and documenting the project alongside his partner Aidan Nunn. Mr. Nickerson has demonstrated his experience in machine learning in a previous project where he constructed and compared 3 different neural networks that could classify a song by its genre. Using his past knowledge, he plans to contribute most towards constructing the Local Logic Module and the Kamiak Logic Module.

Developer Bio: Aidan Nunn is an undergraduate student at Washington State University. He is close to achieving a Bachelor's Degree in Computer Science. Aidan enjoys working with data and optimizing programs. His role in this project is to help plan, develop, and test machine learning programs for Nunnerson Computing. Aidan has previously worked on perceptron classifiers. His skills include C/C++/C#, Python, Haskell, HTML and CSS.

III. System Requirements and Specification

III.1. Project Stakeholders

Our primary stakeholders are the research team in the WSU Department of Psychology Behavioral Neuroscience Lab. This primarily includes Dr. Angela Henricks and her graduate student Kelly Hewitt, but also involves the other members of the lab.

The model we build could be used by future research teams to better understand the link between neural processes in the brain to real life behavior. These potential future researchers are stakeholders as well. Furthermore, if researchers come to understand these mechanisms they could attempt to manipulate them to better help people who suffer from addictive behavior (i.e. drinking alcohol, eating, smoking, etc…), so people helped by this research are potential stakeholders as well.

III.2. Use Cases

Story: Kelly Hewitt is a neuropsychologist at WSU who is researching rodent behavior. Kelly is currently studying four groups of rodents: non-alcohol dependent males, non-alcohol dependent females, alcohol dependent males, and alcohol dependent females. For each group of rodents, Kelly measures the local field potentials in multiple parts of the rodent's brain. After

collecting the data, Kelly would like to analyze the LFP values measured when the rodent was in the decision-making process of drinking alcohol. To do this, Kelly uses our software.

Kelly uploads a previously constructed machine learning model into the software. Kelly uses the software to apply the newly generated model to her new data. The software returns five second timestamps corresponding to when the rodent was in the decision-making process. Kelly saves the timestamps onto her computer and repeats the same process on the remaining three rodent groups. Kelly then closes the software and begins her analysis of the newly processed data.
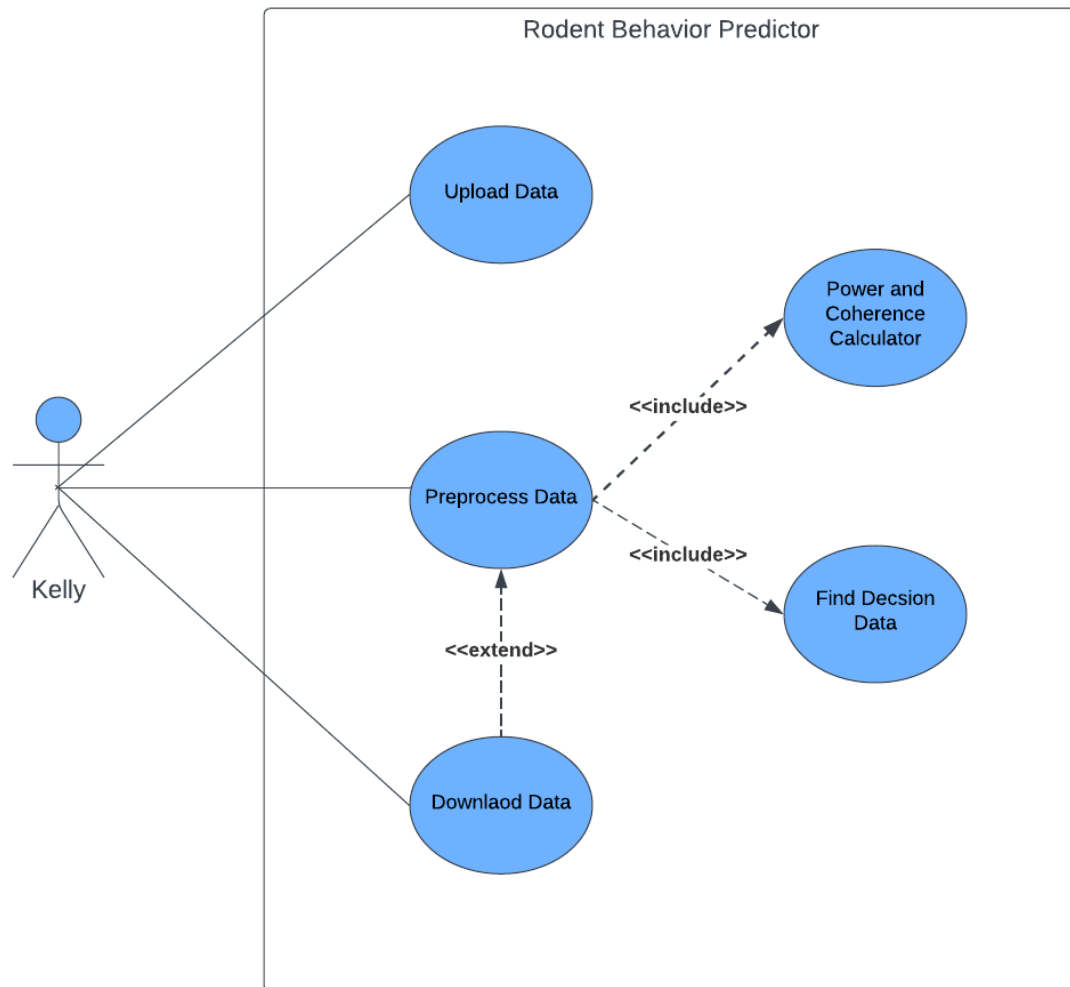
Story: Angela Henricks is a researcher at WSU who is researching rodent behavior. Angela uses a machine learning model to predict the actions of rodents who are dependent on alcohol. However, the model has been giving low accuracy on its predictions and Angela would like to retrain the model using a larger data pool. In order to accurately train the model, Angela needs to process an enormous amount of data, enough that it would take a very long time on a desktop computer. Angela needs to use the Kamiak Cluster to process the data efficiently.

Angela begins by sorting each data file she would like the model to learn from into a file explorer folder. Then, Angela can send her data and our software files to the Kamiak Cluster. Using instructions and terminal commands that we have provided, Angela can request a job

from the Kamiak Cluster. Kamiak runs our software and trains a new machine learning model using Angela's data. Kamiak returns the results of the training.

Story: Angela Henricks is a researcher at WSU who is researching rodent behavior. Angela uses a machine learning model to predict the actions of rodents who are dependent on alcohol. She has recently computed a new model and hopes that it will have better accuracy than the previous one. Angela can use our software to test the accuracy of the new model.

To start, Angela loads the new model into our software. Then, she selects the option to test the accuracy. The software includes preinstalled sample data that can be used in accuracy testing. The preinstalled sample will be small enough that it can be used on a desktop computer, but not too small as to undermine the accuracy of the test. The application will run the model against the sample and return its accuracy. Angela can then decide whether the given accuracy is appropriate and choose whether to rebuild the model using more data or test the model again against more data.

III.3. Functional Requirements

III.3.1. [Machine Learning Model]

Rodent Decision Making Recognition Tool: This machine learning model tool must be able to recognize when a rodent is in the decision making process of consuming alcohol. This model will be tested on female and male rodents that are either addicted to alcohol or are sober. This will be important for knowing when to start and end data collection.
Source: Dr. Hendricks and her team have requested that this be in the project
Priority: Priority Level 0: Essential and required functionality

III.3.2 [Automated Data Collection]

Local Field Potential Data Collection Tool: One the machine learning model recognizes when a rodent is in the decision making process, this tool must begin collecting local field potential data (LFPs) in order to calculate the coherence and power values taking place within the rodent's brain. This is important data for Dr. Hendricks research and is essential to the project
Source: Dr. Hendricks and her team have requested that this be in the project
Priority: Priority Level 0: Essential and required for functionality

III.3.3. [Graphic User Interface]

User Interface for Automated Data Collection of Rodent LFPs: This tool is important for assisting users with using the predictive model to collect data. Users should be able easily import data into the program. This program should then neatly output the LFP data with timestamps of when that data was recorded. This entire process should be intuitive for the user and easily accessible for people who have little experience with coding.
Source: Dr. Hendricks and her team have expressed their desires for having this function in the project.
Priority: Priority Level 2: Extra features or stretch goals

III.3.4 [Power and Coherence Calculator]

Power and Coherence Calculator: This tool will take in the recorded LFP data and convert it into coherence and power values. These values are important because they will be used for testing and training data in our machine learning model.
Source: Dr. Hendricks has informed us that the power and coherence values have been shown to work well for training a predictive model in the past.
Priority: <u>Priority Level 0</u>: Essential and required for functionality

III.3.5 [Interface with the Kamiak Cluster]

Interface with the Kamiak Cluster: This functionality will allow users to easily utilize the Kamiak Cluster for building machine learning models. We will include instructions for using the terminal or programs like PuTTY for accessing Kamiak.
Source: Dr. Henricks has informed us that she would like us to use the Kamiak Cluster to speed up calculations.
Priority: <u>Priority Level 0</u>: Essential and required for functionality

III.4. Non-Functional Requirements

Easy to Use: The final product should be easy to use by future researchers who have little to no knowledge in coding.

Readable Code: Once this project is complete the codebase should be easy to understand. All the code should have comments to describe the functionality of the code so future developers of the product can add to and maintain the code.

Fast: The speed it takes to process the data and calculate the outputs should be relatively fast. To speed up processing time we will be using WSUs Kamiak supercomputer.

Extensible: The code should be easily expandable by future developers who want to add features.

Documented: Directions on how to use functions and features in the project should be available for future users so if they do get confused on how to use the program, they can refer to a guide.

IV. Software Design

IV.1 Architecture Design

IV.1. Overview

Nunnerson Computing has decided on a Client-Server model for this project. We considered using a Model-View-Controller architecture, but since there isn't much need to store data we opted to stick with the Client-Server model instead. We will design a terminal frontend for the researchers to use. The software will also have a local backend that stores the logic for the program. This local backend can be used to run less intensive computations that we don't need Kamiak for. The software will be able to be sent to Kamiak for intensive computations, too.

The entry point for the user is the Front End, which is the Client half of the Client-Server model. The Front End contains the interface where the user will input commands and view results of their requests. Requests will be interpreted by an API Controller module which manages the modules of the program.

The API Controller uses the External Data Access and Preprocessing Module to preprocess data from the files inputted by the user. Then we move to the Local Logic Module. The Local Logic Module stores algorithms for building a machine learning model, testing a machine learning model against sample data or using the model to predict rodent behaviors. If an intensive job is requested, like building a new machine learning model, the software can be uploaded to Kamiak. Our team will provide step-by-step instructions for using our software with Kamiak. Kamiak will run the job and return the results to the user.
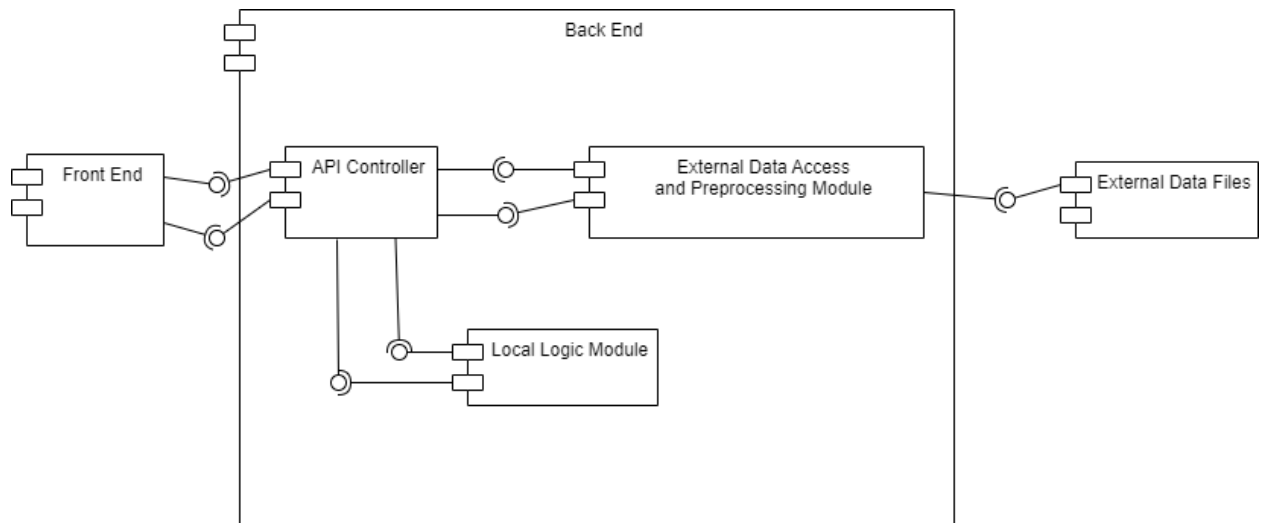
III.2. Subsystem Decomposition

I.1.1. API Controller

    a) Description

The API Controller is the connection between our frontend and the backend of our software. It handles job requests from the frontend and uses the logic module and preprocessing module to perform those jobs, as well as returning the results of those jobs to the frontend.

    b) Concepts and Algorithms Generated

This subsystem will contain three classes: one class with methods to interface with the front end, one class to access the logic module, and one class to interface with the preprocessing module.

    c) Interface Description

Services Provided:

1. Service Name: SendPreprocessingJob
   Service Provided To: Preprocessing Module
   Description: Send a preprocessing job request to the preprocessing module.

2. Service Name: SendLogicJob
   Service Provided To: Local Logic Module
   Description: Send a job to the local logic ,odule

3. Service Name: AcceptJob
   Service Provided To: Front End
   Description: Accepts a job from the frontend and sends it to the logic module or the preprocessing module.

4. Service Name: ReturnOutput
   Service Provided To: Front End
   Description: Sends the results of a job to the frontend.

Services Required:

1. Service Name: ReceiveData
   Service Provided From: Front End

2. Service Name: ReceiveData
   Service Provided From: External Data Access and Preprocessing Module

3. Service Name: SendOutput
   Service Provided From: Local Logic Module

I.1.2. External Data Access and Preprocessing Module

a) Description

The External Data Access and Preprocessing Module accesses user submitted data and preprocesses it for use in the logic module. The interface is made up of a class. This class will contain a private data structure object. This will also contain multiple public functions including loading in data, writing data, power calculators and coherence calculators. This module relies on functions from the Plexon Software Development Kit to access the .pl2 data files used by our client. These functions are accessed via a DLL provided by Plexon.

b) Concepts and Algorithms Generated

The External Data Access and Preprocessing Module subsystem will need read and write methods to load in new LFP data and output new power and coherence values. This subsystem will also need power and coherence calculators. These algorithms will take the inputted LFP data and convert it into power and coherence values. The power and coherence values are vital for training and testing the machine learning model. Once the values are calculated, they will be added into a Pandas Dataframe. Before calculating the power and coherence values, the subsystem cleans the data as well.

c) Interface Description

Services Provided:

1. Service Name: LoadData(File)
   Service Provided To: API Controller
   Description: Read in the external data files and put it into a data structure.

2. Service Name: WriteToFile(Data)
   Service Provided To: External Data Files
   Description: Writes the output data neatly to a CSV file.

3. Service Name: CalculatePower(Data)
   Service Provided To: API Controller
   Description: Takes in a list of LFP values and returns a list of power values.

4. Service Name: CalculateCoherence(Data)
   Service Provided To: API Controller
   Description: Takes in a list of LFP values and returns a list of coherence values.

5. Service Name: CleanNoise(Data)
   Service Provided To: API Controller
   Description: Takes in a list of LFP data and returns a noiseless list of LFP values.

6. Service Name: SendData
   Service Provided To: API Controller

Services Required:

1. Service Name: Send Job

Service Provided From: API Controller

I.1.3. Local Logic Module

a) Description

The Logic Module Local holds algorithms for computations like constructing a model and using the model to test the accuracy of a machine learning model against sample data and using the model to make predictions on input data. If these jobs are too intensive to be run locally, we will use Kamiak to perform those computations by sending our data and software to the Cluster to be run.

b) Concepts and Algorithms Generated

The Local Logic Module has a function that builds a new machine learning model with user submitted data. It can also run a recently built machine learning model or one sent from the API Controller on preprocessed input data to return an output. This output should just contain the start and end timestamps of when the rodent was in the decision making process of consuming alcohol. We will also include functionality to allow a user to test the accuracy of an inputted model against sample data.

c) Interface Description

Services Provided:

1. Service Name: PredictTimestamps()
    Service Provided To: API Controller
    Description: Runs the predictive model on preprocessed data and returns timestamps of when the rodents are consuming alcohol.

2. Service Name: TestAccuracy
    Service Provided To: API Controller
    Description: Tests an inputted machine learning model against sample data and returns the accuracy.

Services Required:

1. Service Name: SendData
    Service Provided From: External Data Access and Preprocessing Module

IV. Data Design

We don't expect this project to require permanent data structures other than one to hold sample data for testing the accuracy of a machine learning model. Data and models will be stored in external data files that will be uploaded to the program. As for temporary data structures, we believe that using arrays to store LFP data will be sufficient. An array will also be used to store the sample testing data. Pandas dataframes will be used to store processed data for use in Lasso.

V. User Interface Design

The GUI will consist of a terminal window with a list of available commands. Each of these commands will have a progress bar showing how far along the command is. Text output will be displayed to the terminal window. We will use MatPlotLib to plot graphs, so graphs will be displayed in their own windows using the functionality of that library. Additionally, file operations like choosing a directory to retrieve data from will be performed using tkinter's filedialog() function.

VI. Testing Plan

VI.1. Test Objectives and Schedule

The Nunnerson Computing team will use testing to see how reliable our application is. We will write test cases that probe our system for potential weak points that need to be addressed in the future. After fixing all the issues found in testing we will have ensured our applications functions as it should. Additionally future changes to the application should be easily testable using the robust test functions we coded.

Nunnerson computing will utilize the PyUnit testing framework along with the unittest library to test the functionality of the system and its parts. For each working function written, a team member will write a set unit tests to probe where the function could fail. Once all the unit tests have been passed only then can the functional code be merged to the main branch. Larger functions that depend on lower level functions should be tested. To find these larger integration functions, a call graph will be made to help us find what functions are needed for integration testing.

There will be two deliverables for the testing and development process. Our GitHub repository will contain a folder with all the testing functions. The test functions will be separated by class type for each test file. We'll also have documentation for how to run tests. The milestones in our testing process are: Write code, write unit tests, document and fix issues found, write integration tests, document and fix issues found from integration tests and merge. These milestones could be achieved multiple times in our development lifecycle. Any bugs that are documented during the testing period should be added to GitHub as an issue. Once the bug has passed testing, only then can the function be merged to main.

VI.2. Scope

This section contains a general description of our testing plans for our Python machine learning program. It will discuss our testing strategy and how we intend to implement each type of testing on our program, as well as the environment requirements necessary to run our tests.

VI.3. Testing Strategy

Our team has never been tasked with testing machine learning before, so we plan on focusing testing on types of components we have previous experience writing tests for and researching the best ways to test machine learning later on. Testing the machine learning part of our project will involve testing the accuracy of a generated model against sample data instead of writing unit tests to compare the generated output to an expected output. Additionally, testing machine

learning in this project is made more difficult because of the fact that we will be running our program remotely on the Kamiak Cluster, so it will take some time to run the entire program once we reach system testing. We intend on using Continuous Integration in our testing. This is because once our algorithm is set up, most of the work will involve fine-tuning for speed and accuracy, so continuous merging will allow developers to be on the same page with each other on what kind of results their tuning has and allow positive changes to reach all developers quickly. Our process for testing other components is outlined below:

A. **Write Code:** We need to write code before we can test it! The nature of machine learning will be to establish a protocol for building a model then optimizing that protocol for speed and accuracy. Code will include implementations of the Kamiak Logic Module, API Controller, Local Logic Module, External Data Access and Preprocessing Module, and Front End.

B. **Identify Testable Features:** Since the main function of the code can't be tested traditionally with unit tests, we will identify methods that can be unit tested as well developing as a call graph for integration testing.

C. **Determine Testing Methodologies:** Depending on the feature being tested, we might want to employ Blackbox or Whitebox testing to ensure thorough test coverage.

D. **Identify Test Inputs and Oracles:** We will examine each feature that needs testing and determine inputs and oracles based on the feature's requirement specifications.

E. **Write Tests:** We will write the tests for the selected features.

F. **Run Tests:** We will run the tests for the selected features.

G. **Record Test Results:** We will record the results of tests, successful or not, and include them in our revision of this report.

H. **Document and Fix Errors, Then Retest:** In the case of errors, first we will document them as an issue on GitHub, then claim that issue and fix the error. The feature needs to pass retesting in order for the issue to be closed.

I. **Merge:** Once a feature has passed the tests, it can be merged to our main branch. A developer will create a merge request on the project Gitlab repository and accept the request.

VI.4. Test Plans

VI.4.1 Unit Testing

We will use PyUnit and its mocking library for unit testing. We will test all non-trivial methods with one passing input and one failing input. Blackbox testing may be utilized in the case of more complex methods where we want to ensure more complete input coverage. Trivial methods involve empty methods or those who provide a very simple function, like comparing two characters, whose accuracy can be verified visually. Non-trivial functions are any used often (more than once) or whose function is very essential to the core operation of our program. Methods that might require blackbox testing are those with large input variability, for example public functions that accept user input.

VI.4.2 Integration Testing

PyUnit and its mocking library will also be used for integration testing. First, we will make a call graph for each component of the program, then a broader call graph that connects each component. Each component will be tested via top-down, where dependencies are mocked in

order to allow for integrating one unit at a time. We would like to run integration tests on a class by class basis first. Functions that depend on smaller component functions should be tested to see if the method meets the expected results. If an error is found during integration testing, the error should be identified, documented, fixed, and retested. Once the function passes all the integration tests, mark testing issues as complete in GitHub and merge the tested function to the main branch.

VI.4.3 System Testing

System testing will be difficult since the output of our program is stochastic. The following sections are filled in regardless, however we expect this stage of testing to involve slowly optimizing our algorithm for speed and accuracy rather than testing for expected oracles like one would do for other software projects. Testing for better accuracy can be done by playing around with the testing and training size of the datasets. Finding a good middle ground for the test and training set could help with solving overfitting and underfitting. The total size of the data being used on the model should be tested for performance too. Too much data could slow down the system because the model is under immense computational stress. Too little data could cause the machine learning model to be inaccurate.

VI.4.3.1 Functional Testing

Our functional testing plan will involve looking back to the requirements specifications laid out by our client and ensuring that their requests are met. So far our client has given loose requirements: use the Kamiak Cluster to build a machine learning model that identifies when a test rodent is about to self administer alcohol. We will of course expand our suite of functional tests if given more requirements in the future, but currently as long as the requirements outlined above are met we will be satisfied.

VI.4.3.2 Performance Testing

Performance testing will involve testing our algorithm for speed and accuracy. Mainly we will be testing the limits of our algorithm in how much data can be reasonably processed. We will also run our program on different amounts of data to figure out how much we need to prevent overfitting. Currently it's common practice to split training and testing sets in 80% data in training and 20% data in testing. We will test out different ratios to determine which gives the best accuracy without overfitting and underfitting the model. The amount of data being used for the train and test sets should be probed for efficiency as well. Additionally, performance testing will involve testing how accurate the program is by using different configurations of input data and lambdas for lasso.

VI.4.3.3 User Acceptance Testing

User acceptance testing will be done to ensure our team has fulfilled the requirements demanded by our client. Each demo will give our client the opportunity to critique functional specifications and/or request new ones. This will give us confirmation on what works and doesn't work in the application. During the demo, our team will have our client perform a list of activities which will showcase the functionality of the application. This list will be based on the Requirements and Specifications section of this document. These lists will refresh the client on the requirements they wanted in the application. For each demo, any bug that is found by the client should be documented by one of the developers and posted as an issue to the Github

repository. At the end of the demo our team will take time to discuss with our client about what they liked and disliked about the demo and work together to decide what to do next.

VI.5 Environment Requirements

Our testing environment will be using the PyUnit testing framework. Tests will be separated based on class type and stored in their own test files. Each file will have a method for automatically running the test cases for each function. If an error arises the tests will print what went wrong to the terminal. We will also need to create tests for integrating Kamiak. This is important because we want to see how well our model works with a large dataset. The entire testing environment should be stored in a folder on GitHub called TestingEnvironment.

VI.6 Test Results

Preprocessing Module Tests

>   Method Under Testing: splitSignal()
>   Aspect Being Tested: This function allows the code to split a signal into traditional frequencies used in studying neurobiology
>   Expected Results: The method will return an array matching [6.5, 4, 6, 11, 13, 5]
>   Observed Results: The method returned an array matching [6.5, 4, 6, 11, 13, 5]
>   Test Results: The tests passed its unit test.
>   Test Case Requirements: The length of the inputted arrays must be the same length and the numbers in the array must be positive.

>   Method Under Testing: CreateHeader()
>   Aspect Being Tested: This function creates an array of headers for the power and coherence values.
>   Expected Results: The method will return an array matching all the power and coherence headers.
>   Observed Results: The method returned an array matching all the power and coherence headers.
>   Test Results: The test passed.
>   Test Case Requirements:

>   Method Under Testing: getFileNames()
>   Aspect Being Tested: This function gets the file name of all the files in a directory. This method reads a directory on your local PC.
>   Expected Results: The method will return an array matching ['A170_post0_2020-4-20_pl2_spl_ead_plx.pl2]
>   Observed Results: The method returned an array matching ['A170_post0_2020-4-20_pl2_spl_ead_plx.pl2]
>   Test Results: The test passed.
>   Test Case Requirements: The file specified in this function must match an existing file location on your PC. Additionally it must also contain .pl2 files with LFP data.

>   Method Under Testing: voltsToRawAD()
>   Aspect Being Tested: This function converts volts into raw analog/digital data.
>   Expected Results: The method will return an array matching [-36, 17, -30, -77]
>   Observed Results: The method returned an array matching [-36, 17, -30, -77]
>   Test Results: The test passed.

Test Case Requirements: The input array must be an array of numbers with a length of 5.

Method Under Testing: noiseArtifactsFilter()
Aspect Being Tested: This function filters out an array of numbers based on a 1.5 threshold, 1 onset, and 2 offset.
Expected Results: The method will return an array matching [0.76, 0.01, 0.04, 0.04, -1, -4, 0.05]
Observed Results: The method returned an array matching [0.76, 0.01, 0.04, 0.04, -1, -4, 0.05]
Test Results: The test passed.
Test Case Requirements: The values for the offset, threshold and onsets must be numbers.

Method Under Testing: downSampling()
Aspect Being Tested: This function takes in an array of numbers and downsamples the array by a rate of 5.
Expected Results: The method will return an array matching
 [0.9523711684276455, 5.966882643370076, 10.815006002654167, 15.909782205158153, 20.67519282567084, 25.839229480847393]
Observed Results: The method returned an array matching
[0.9523711684276455, 5.966882643370076, 10.815006002654167, 15.909782205158153, 20.67519282567084, 25.839229480847393]
Test Results: The test passed.
Test Case Requirements:

Method Under Testing: voltsToRawAD()
Aspect Being Tested: This function converts volts into raw analog/digital data.
Expected Results: The method will return an array matching [-36, 17, -30, -77]
Observed Results: The method returned an array matching [-36, 17, -30, -77]
Test Results: The test passed.
Test Case Requirements: The input array must be an array of numbers with a length of 5.

Local Logic Module

Method Under Testing: LocalLogicModule
Aspect Being Tested: We are testing the accuracy of the model by passing in a variety of different lambda values
Expected Results: There is no expected result. We are testing for accuracy
Observed Results: The highest observed accuracy was the model build with a lambda value of 0.0001
Test Case Requirements: The input array but be an array of positive valued numbers.

Stochastic Testing Of a Binary Model to Predict Whether a Rodent is Exposed to Room Air or Vaporized Alcohol

For our first build of a binary model for predicting if a rodent subject was routinely exposed to room air or vaporized alcohol, we processed data from a batch given to us by our client into one dataframe including only female rats, one dataframe including only male rats, and one

dataframe including all rats in the batch. These models were tested on a lambda value of 0.01. Each run built 100 unique models trained on the same dataset.

We struggled to fit the data to a model. Runs were frequently met with warnings stating that the model would not converge. An example of warnings can be seen in Appendix C, section C4. In order for the models to converge, we increased the fit iterations to 10,000,000. This means the model would try to fit the data a maximum of 10,000,000 times. This setting allowed the model using all data from the batch to converge in each build, but even the models using only male or female data would still struggle to converge.

In addition to struggling to fit, the models produced showed poor accuracy scores. We used Coefficient of Determination, or R2 score, to test the accuracy of our models. This metric was chosen because its results are bounded by 0 and 1, which makes it easily interpretable and explainable as a "0%-100%" accuracy. As seen on the graphs in Appendix C, sections C1, C2, and C3, the accuracy either rode just above 0 or dramatically fell into the negatives. A score below 0.3 means the model provides poor correlation, and a negative score means the model fits the data very poorly or not at all.

These initial results have led us to believe that these runs did not have enough data present to find a pattern. We are expecting another set of data soon, so we will be able to test our theory once that arrives. Our client's reaction to our findings was an agreement that we don't have enough data, especially since they are in the process of preparing more for us. Additionally, they proposed that we try processing the data into power and coherence values across five-second intervals. Currently, we are using the average power and coherence values across an entire testing session. The technique proposed by our client would essentially expand on our limited data and allow the model to find patterns among more data points.

<u>Lambda Testing on the Continuous Model that Predicts the Amount of Alcohol Consumed by the Rodents</u>

For our continuous predictor model, we tested different lambda on multiple models to see which lambda values worked best with the data. Lambda values are values that are passed through the model that determine the learning rate. So far we have managed to complete lambda testing on a continuous model that was trained on both male and female rodents. Our results seen at appendix C5 show that the best lambda values for male, female, and all sexes were, 0.5, 0.2, and 0.1 respectively. The resulting R squared value indicates that our continuous models are not making accurate predictions.

Additionally we only have done testing between sexes and have not separated the data based on experimental and control groups. Lambda testing still needs to be done on models trained with only control male subjects, control female subjects, experimental male subjects and experimental female subjects.

VII. Description of Final Prototype

Our final prototype for this project will follow the Client-Server architecture model. The frontend will be terminal operated by the researchers while the backend will store the logic for the program. There will also be a local backend that will run less intensive programs that would be unnecessary for Kamiak to run. The front end of the program terminal is where the user will input commands to build models and view the results. Users will be able to build the binary

predictor for determining rodent alcohol dependency, and a continuous predictor that determines the amount of alcohol consumed by the rodents. The user will be able to see the accuracies of the models they build. The user will also be able to upload their data and the program to Kamiak in order to process data quickly. In future versions of this document we will include a user manual in this section which describes how to operate the software in detail.

VIII. Projects and Tools Used

Tool/Library/Framework	Quick note on what it was for
os	Used to extract the filename that contains all the LFP plexon files.
pandas	This library was used to organize the data into data frames.
scipy/signal	Used to extract the coherence and power values from the LFP data.
matplotlib/pyplot	Used to graph coherence values based on frequency and slo graphs the mean squared error based on which lambda values was used to train the lasso model.
pypl2/pl2_ad	Used to extract information from individual channels.
sys	Used to get the filename that contains all the plexon LFP files if the filename was passed through the command line.
itertools/combinations	Used to organize the channel data into pairs that can be used to calculate coherence values.
statistics/fmean	Used to take the mean of the frequency bands during data preprocessing
csv	Used for reading and writing to csv files.
sklearn.metric/mean_squared_error	Used to calculate the mean squared error the lasso model
sklearn.model_selection/train_test_split	Splits the power and coherence data into training and testing sets for the lasso model
pypl2/pl2_info	Used to extract the channel information from the .pl2 files.
sklearn.linear_model/Lasso	This library was used to build the lasso

	machine learning model.
numpy	Used to create data structures for the data

Language Used in Project
Python

IX. Product Delivery Status

Our project will be delivered to our client on April 28th, 2023. Our deliverables will include the following:

- Github Code Repository
- Instructions on how to use the software locally
- Instructions on how to use the software with Kamiak
- Details on the proper Python environment to run this software in, including outside packages and DLLs
- This document

Currently, these items can be found in our team's Github Repository. We will demo our software to our clients on our laptop and one of their lab computers as well.

X. Conclusions and Future Work

X.1. Limitations and Recommendations

The current limitations of our project are that it does not include a modern GUI or support machine learning models other than what we currently support. Additionally, it requires the user to manually upload the code and data to Kamiak, despite our original wishes to automate this feature. Implementing a modern GUI is possible, but due to our client's wishes we will not be implementing one since they prefer that we ensure the quality of the backend components first. This is a stretch goal that could be completed if a future team working with this code has extra time. Automating the use of Kamiak would require skills that our team has realized are beyond our scope. While we believe we could find a solution given enough time, due to time constraints we have not researched how to solve this problem and are instead focusing on more critical components of the project. We will be able to manually upload our code and data to Kamiak like the previous team has done, which is sufficient for our client.

X.2. Future Work

Future work for this project would be to fix the limitations of the project. Our client has expressed that they do not want a GUI and would rather have code applicable to their research, but we believe that a GUI would be useful since it would improve the accessibility of the code. Adding more processes to the logic module would also allow for more types of models to be produced.

XI. Acknowledgements

Thank you to our client Dr. Angela Henricks for proposing this project to the Computer Science capstone course and giving our team the opportunity to work on it. We also thank our mentor Ananth Jillepalli for assistance navigating this course and the new challenges it has brought us.

XII. Glossary

API: Application Programming Interface. A way for two or more computer programs to communicate with each other.

Backend: Server-side of an application. The part that the user can't see and interacts with indirectly..

Central Nucleus of the Amygdala: A location in the brain that is responsible for mediating many aspects of fear and anxiety.

Client-Server Architecture: A software design pattern with two main components, a client and a server, where the client makes service requests and the client provides the services.

Coherence: Coherence is the correlation in activity in each site of the brain.

CSV File: A text file which stores data as comma separated values (CSV).

EtOH: Ethanol

Frontend: Client-side of an application. The part the user sees and directly interacts with.

GUI: General User Interface. Standard terminology for a user interface that uses windows, icons, and menus to carry out user commands.

Kamiak: A high performance computer on the Washington State University campus. Built out of a cluster of local computers connected via a high speed network.

Lasso: A machine learning algorithm used to construct a machine learning model.

Local Field Potential (LFP): The result of electrical activity in cells (mainly neurons).

Machine Learning: The use of computer programs that can learn based on input data and make predictions based on future data.

Machine Learning Model: A data file that has been trained to recognize specific data patterns.

Matlab: A high level programming language commonly used by mathematicians and engineers.

Medical Prefrontal Cortex: A region in the brain responsible for planning complex cognitive behavior, and decision making.

Nucleus Accumbans: A major component of the ventral striatum and has long been thought to be a key structure involved in mediating motivational processes.

Power: Power is the measure of each signal in frequency bands.

PuTTY: An SSH client for Windows that can be used to connect to the Kamiak Cluster.

Python: An open-source, high level programming language with many libraries useful for machine learning.

PyUnit: A popular integration testing framework for Python code.

SF: Sweet Fat

Sprague-Dawley Rats: A species of rats ubiquitous to biomedical research.

SSH: Secure Shell is a network communication protocol that allows two computers to communicate and share data.

Stochastic: Having a pattern that can be analyzed statistically but not predicted.

XIII. References

[1] György Buzsáki, Costas A. Anastassiou, and Christof Koch, "The origin of extracellular fields and currents — EEG, ECoG, LFP and spikes," *Nat Rev Neurosci*, June, 2016. [Accessed Sep. 20, 2022]

[2] Lucas L. Dwiel, Jibran Y. Khokhar, Michael A. Connerney, Alan I. Green, Wilder T. Doucette, "Finding the balance between model complexity and performance: Using ventral striatal oscillations to classify feeding behavior in rats," *PLOS Computational Biology*, April, 2019. [Accessed Sep. 20, 2022]

[3] Wilder T. Doucette, Lucas Dwiel, Jared E. Boyce, Amanda A. Simon, Jibran Y. Khokhar, Alan I. Green, "Machine learning based classifications of deep brain stimulation outcomes in a rat model of binge eating using ventral striatal oscillations," *Frontiers in Psychiatry*, August, 2018. [Accessed Sep. 20, 2022]

[4] "Kamiak cheat sheet," *Washington State University*, October, 2021. [Accessed Nov. 9, 2022]

[5] Aurora Clark, Peter Mills, Jeff White, "Welcome to Kamiak," *Washington State University*, February, 2018. [Accessed Nov. 9, 2022]

XIV. Appendix A - Team Information

Charlie Nickerson   Aidan Nunn

MVP Project Report   22

charles.nickerson@wsu.edu   aidan.nunn@wsu.edu

XV. Appendix B - Project Management

Our team met with our clients once per week during their Monday lab meetings. Our team had the opportunity to explain our current progress to our client and ask clarifying questions about their expectations and their data.

We used multiple tools throughout our project. Our most essential tool was Github. This allowed our team to organize our work by using Github Issues, Milestones, and the Project Board functionality. Our team communicated using messaging apps like Discord and Snapchat. We used Discord for voice chatting to discuss sprints and project goals. Zoom was used to communicate with our mentor and clients.
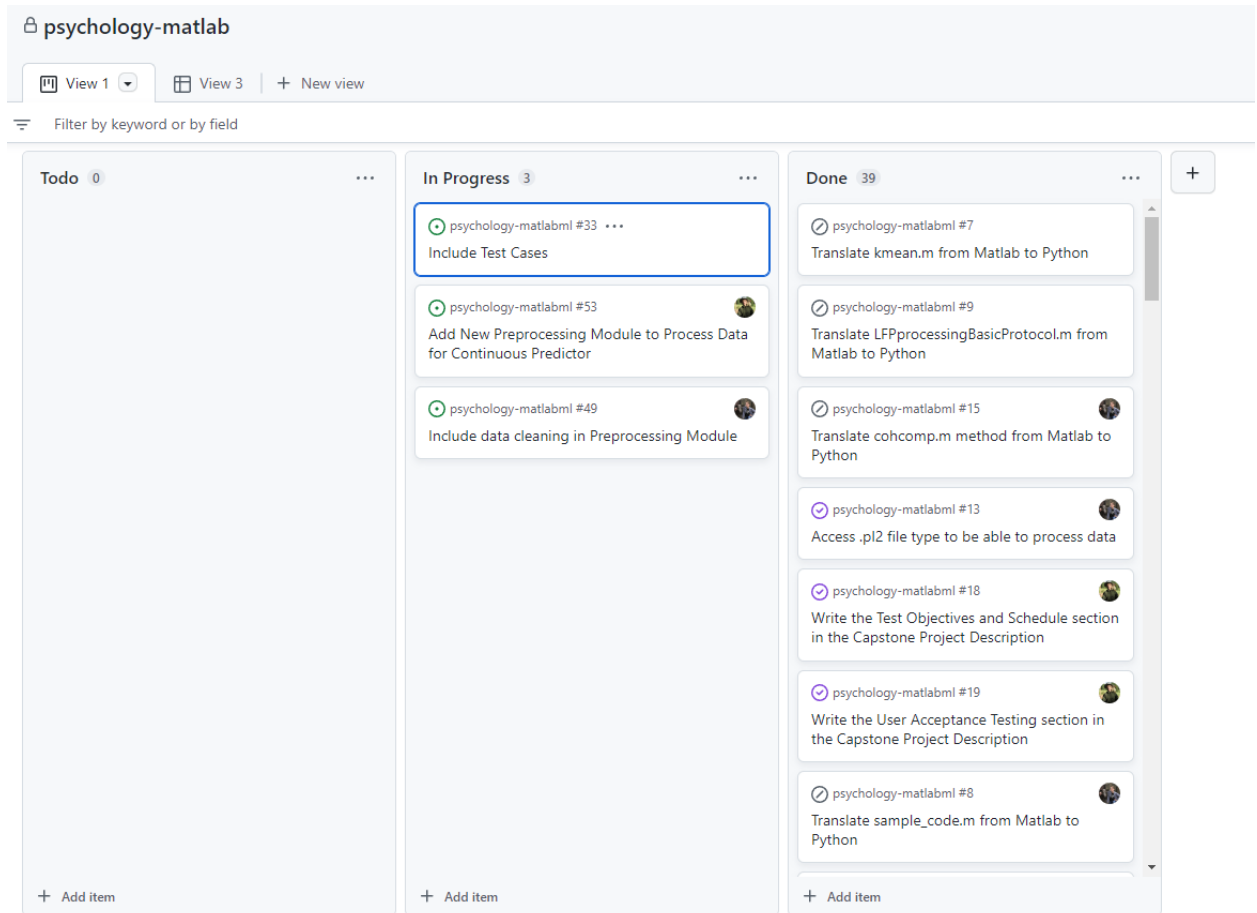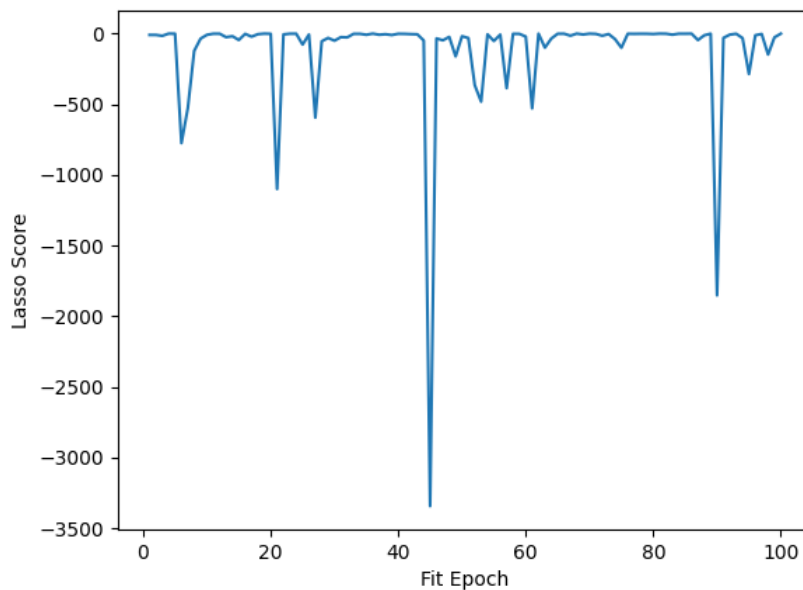

Example of our Github Project Board.

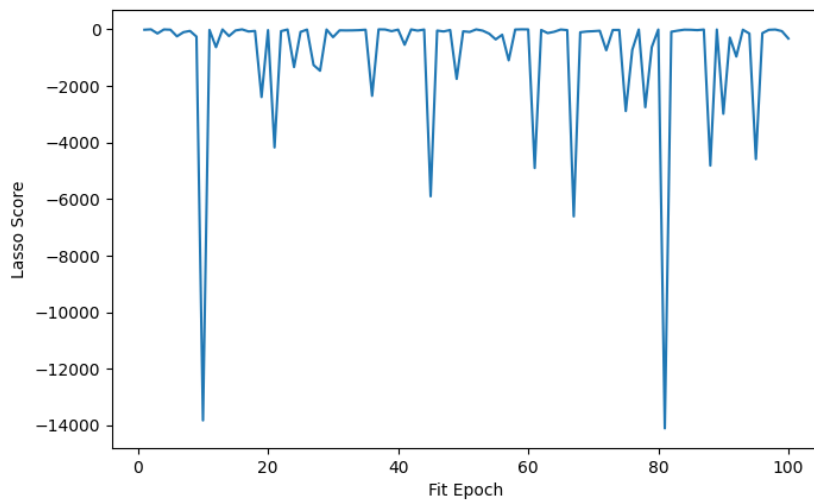# XVI. Appendix C - Testing Results

XVI. Appendix C1
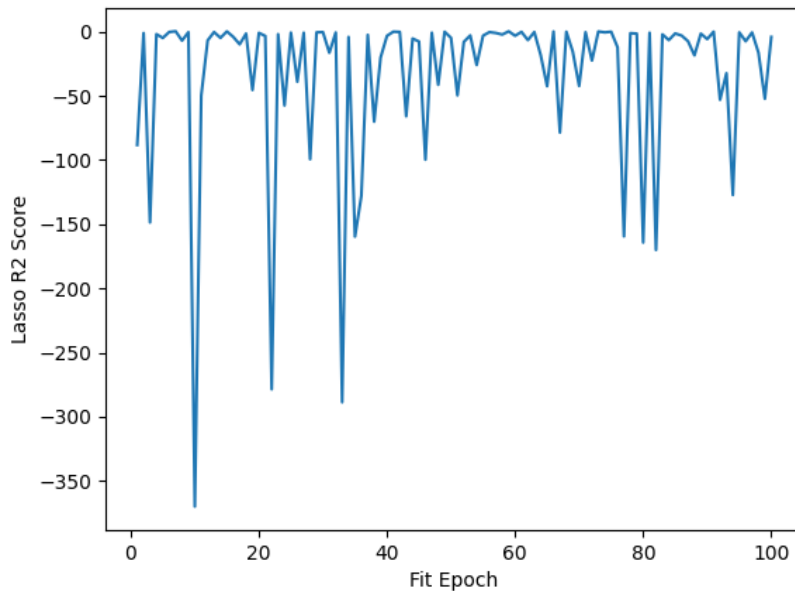
This graph shows the R2 lasso scores of 100 different binary room air vs vapor models trained with both male and female subjects.

XVI. Appendix C2

This graph shows the R2 lasso scores of 100 different binary room air vs vapor models trained with only female subjects.

XVI. Appendix C3

This graph shows the R2 scores of 100 different binary room air vs vapor models trained with only the male subjects.

XVI. Appendix C4

```
On Epoch 0 of 100
Building Room Air vs Vapor Model
Splitting Data
Fitting Data
Accuracy: -1.3996625170007446
On Epoch 1 of 100
Building Room Air vs Vapor Model
Splitting Data
Fitting Data
C:\Users\aidan.nunn\Documents\GitHub\Capstone\psychology-matlabml\pythonProject\venv\Lib\site-packages\sklearn
 \linear_model\_coordinate_descent.py:648: ConvergenceWarning: Objective did not converge. You might want to
 increase the number of iterations, check the scale of the features or consider increasing regularisation.
 Duality gap: 8.420e-04, tolerance: 4.950e-04
   model = cd_fast.enet_coordinate_descent(
Accuracy: -100.27316269441317
On Epoch 2 of 100
Building Room Air vs Vapor Model
Splitting Data
Fitting Data
C:\Users\aidan.nunn\Documents\GitHub\Capstone\psychology-matlabml\pythonProject\venv\Lib\site-packages\sklearn
 \linear_model\_coordinate_descent.py:648: ConvergenceWarning: Objective did not converge. You might want to
 increase the number of iterations, check the scale of the features or consider increasing regularisation.
 Duality gap: 8.245e-04, tolerance: 4.950e-04
   model = cd_fast.enet_coordinate_descent(
Accuracy: -7.323380273718495
On Epoch 3 of 100
Building Room Air vs Vapor Model
Splitting Data
Fitting Data
C:\Users\aidan.nunn\Documents\GitHub\Capstone\psychology-matlabml\pythonProject\venv\Lib\site-packages\sklearn
 \linear_model\_coordinate_descent.py:648: ConvergenceWarning: Objective did not converge. You might want to
 increase the number of iterations, check the scale of the features or consider increasing regularisation.
 Duality gap: 2.237e-03, tolerance: 4.950e-04
   model = cd_fast.enet_coordinate_descent(
Accuracy: -46.697884811144235
```
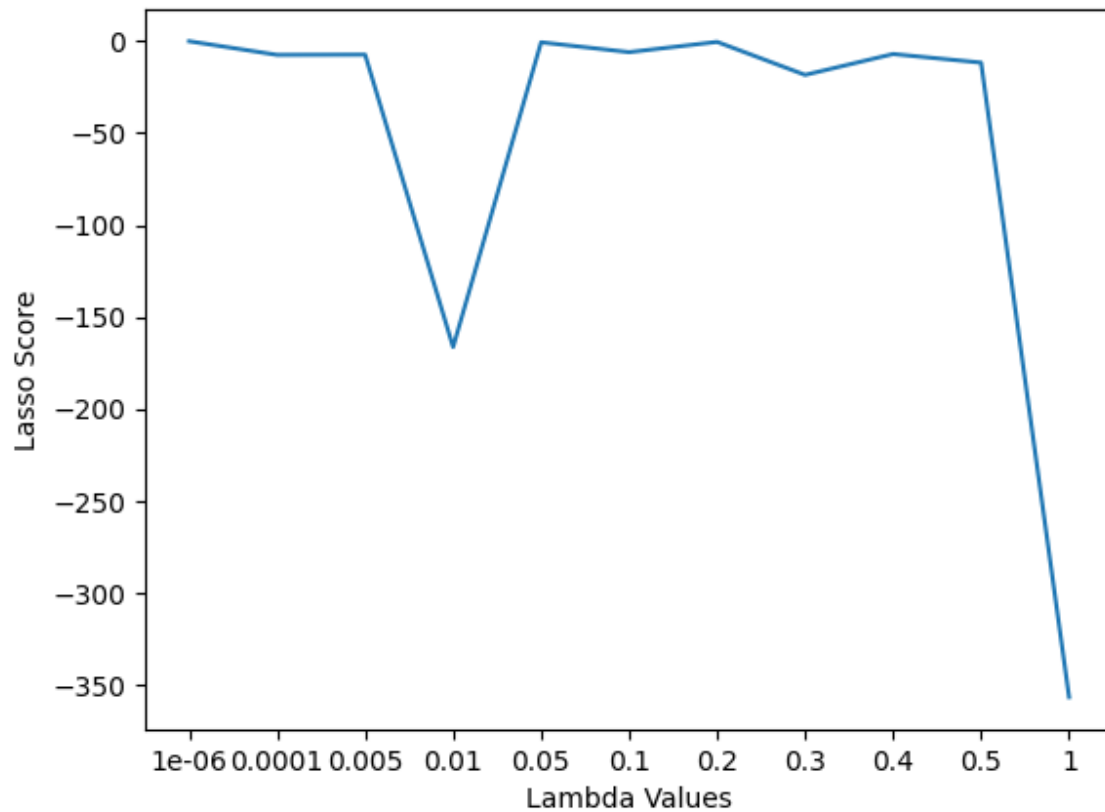
This image shows that our binary models are not converging with the amount of data that was given to us.
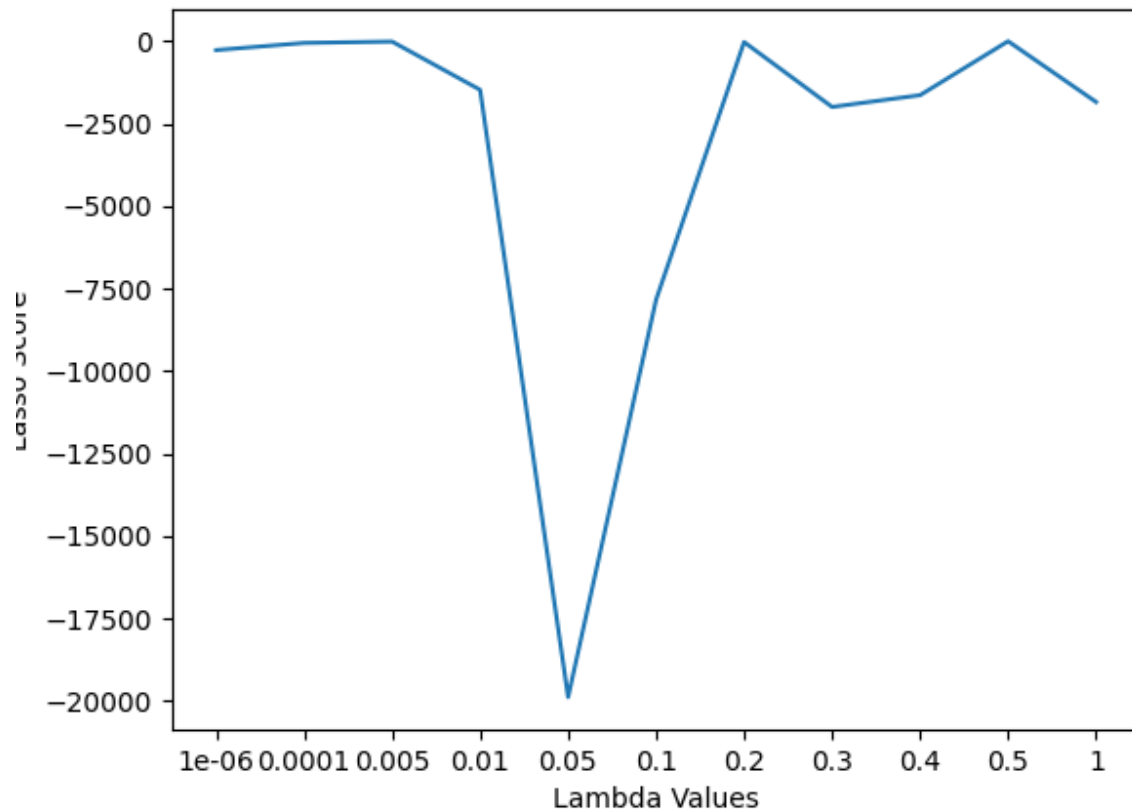
XVI. Appendix C5
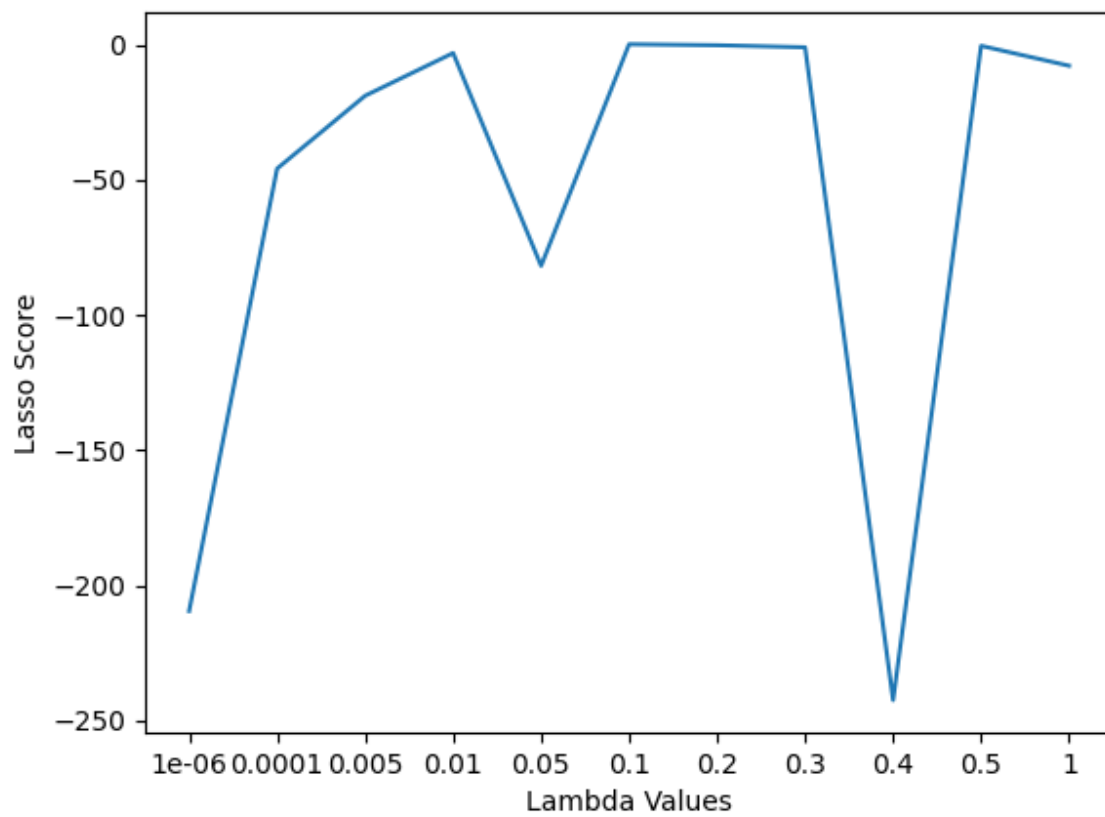
This graphs shows the R2 accuracies of the continuous predictor over all the rats with different lambda values.

This graphs shows the R2 accuracies of the continuous predictor over all the female rats with different lambda values.

This graphs shows the R2 accuracies of the continuous predictor over all the male rats with different lambda values.