Machine Learning Approach to Identifying Neural Features That Predict Rodent Behavior

Washington State University - Department of Psychology

Nunnerson Computing

Aidan Nunn, Charlie Nickerson
9/21/22

TABLE OF CONTENTS

I. Introduction

This document showcases Nunnerson Computing's knowledge thus far regarding our partnership with Dr. Angela Henricks' team and the WSU Department of Psychology. We are assisting Dr. Henricks by developing a machine learning model using lasso (least absolute shrinkage and selection operator) which will predict when to collect data from their Sprague-Dawley rodent test subjects. Their previous method of data collection has been time consuming, and by creating this machine learning model we will streamline the data collection process and save time for the researchers. Our end goal is to present Dr. Henricks' team with a machine learning model which can predict what frames of data they need to study, thereby removing the need for them to manually locate desirable frames of data.

II. Background and Related Work

Among individuals who practice substance abuse, there seem to be neurological processes that underlie behaviors linked to the abuse of those substances. Dr. Henricks and her team hypothesize that LFPs could be used to determine whether substance abuse was occurring or if the male and female subjects were partaking in normal activity.

Currently, the focus of Dr. Henricks' team is on the difference between males and females concerning which parts of the brain are responsible for the decision making process to self administer alcohol. Dr. Henricks' team is not the first to tackle psychology related questions by using rat test subjects. Doucette et al. used the lasso (least absolute shrinkage and selection operator) algorithm with LFP data to predict outcomes of brain stimulation on a rat's desire to binge eat [3]. Dwiel et al. used lasso to predict the amount of food eaten, the increase in consumption following food deprivation, and the type of food eaten by the rats [2]. Additionally, Dwiel et al. managed to predict if a rat was going to consume 42.5 seconds before it would happen [2]. These and previous studies have used male models, however, which work well on other male subjects but do not translate over to female subjects. Due to this, Dr. Henricks' team suspects that a different part of the brain is responsible for the decision to self administer in females and are making female brains a priority in their research.

The main method for gathering behavioral data in experiment subjects, Sprague-Dawley rats, are ventral striatal local field potentials (LFPs). A previous experiment conducted by Dr. Hendricks and her team found that LFP measurements taken from specific regions in the brain can indeed be used in a machine learning model to differentiate LFPs recorded during the consumption of ethanol (EtOH) and sweet-fat foods (SF) from all other behaviors. This is great news, as it means Hendriks' team is already familiar with using machine learning in their experiments. Our machine learning model will still be making new contributions to the project domain, as it's assisting streamline their research.

We need to learn new technical knowledge and skills in order to accomplish this project. Most importantly is understanding the LFP data collected from the rats and understanding the MatLab programming language for use in constructing the machine learning program that will construct our model. We will also need a thorough understanding of the lasso machine learning algorithm. In order to aid us in learning more about the background of this project, Dr. Henricks provided our team with some papers covering previous research that her team referenced when working on their research. These were helpful for understanding how the LFP data was collected from the rats but it is still unclear how the code was implemented to make a predictive model. Later on Hendricks will provide us with the codebase from previous capstone projects. These will be helpful for teaching us how to use MatLab and lasso.

III. Project Overview

Dr. Henricks' and her team are studying which parts of the brain are involved in the decision making process to self administer addictive substances. Their experimental process involves placing a rat in a box for 30 minutes and recording the local field potentials (LFPs) using electrodes in the rat's brain. Inside the box is a lever that dispenses alcohol, and a camera that records the rat for the entire session. Currently, the team watches the video footage and waits for the moment when the rat is making the decision to self-administer alcohol or not; this is when they want to study the LFPs. Our goal is to improve upon a machine learning program which will generate a model that predicts when the rat is making a decision. With this model, the team can isolate the frames of data collected from when the rat is making the decision whether or not to administer alcohol. This will allow them to save time reviewing experiment sessions and avoid accidentally reading data collected from when the rat is not in the decision making process.

Our team has been recommended to use the lasso (least absolute shrinkage and selection operator) algorithm to generate our model. We will use LFP data collected from experiment sessions using plex to train the model. Lasso is a regression analyst method that calculates regularization and variable selection in order to increase prediction accuracy and help with the interpretability of the regression model. Lasso was originally created for linear regression models, but has other generalizations as well including generalized linear models, generalized estimating equations, proportional hazards models, and M-estimators. Lasso works by forcing the sum of the absolute values of the regression coefficients to be less than a certain value. By doing this some coefficients are forced to be zero and will be excluded from impacting the prediction. The lasso method is similar to ridge regression because it shrinks the size of coefficients but it does not set any coefficients to zero making the method unable to perform variable selection. Doucette et al. used the MatLab package *Glmnet* to assist in constructing their model [3], so this is something we will look into as well.

Based on the work performed by last semester's computer science capstone team, Dr. Henricks has recommended that we spend our first semester on the project learning about their research and studying the codebase left behind by the previous students. We will also have the opportunity to play with small LFP datasets and learn lasso by constructing prototypes of our project. This time period will also involve planning out our second semester on the project. Our team's second semester will involve coding our algorithm and deriving an accurate model that will be used by Dr. Henricks' team. At this point we will also begin presenting our findings to the Hendricks lab and potentially to conferences such as SURCA, as outlined in the abstract provided to our team.

We will be utilizing a node in the Kamiak computing cluster on the WSU campus to run our algorithm. We will be working with a great deal of data in order for our model to be accurate for many scenarios, the additional computer power of Kamiak will ensure that we can create test models without wasting too much time. The previous round of capstone students assisting Dr. Henricks created a protocol for using Kamiak. They used Python while working on this problem; however, Dr. Henricks has recommended we try using MatLab instead, as it runs faster than Python and she has a better understanding of that language. MatLab has built in tools for data analysis, like the aforementioned *Glmnet*, and tools for graphics that will help us present our data.

IV. Client and Stakeholder Identification and Preferences

Our client is the WSU Department of Psychology with Dr. Angela Henricks as our primary contact for the project. The model we build will be used by Dr. Henricks' team, but could potentially be used by other teams if they take over the research. Other stakeholders include the rest of Dr. Henrick's team. Our team mentor is Ananth Jillepalli, an assistant professor at WSU.

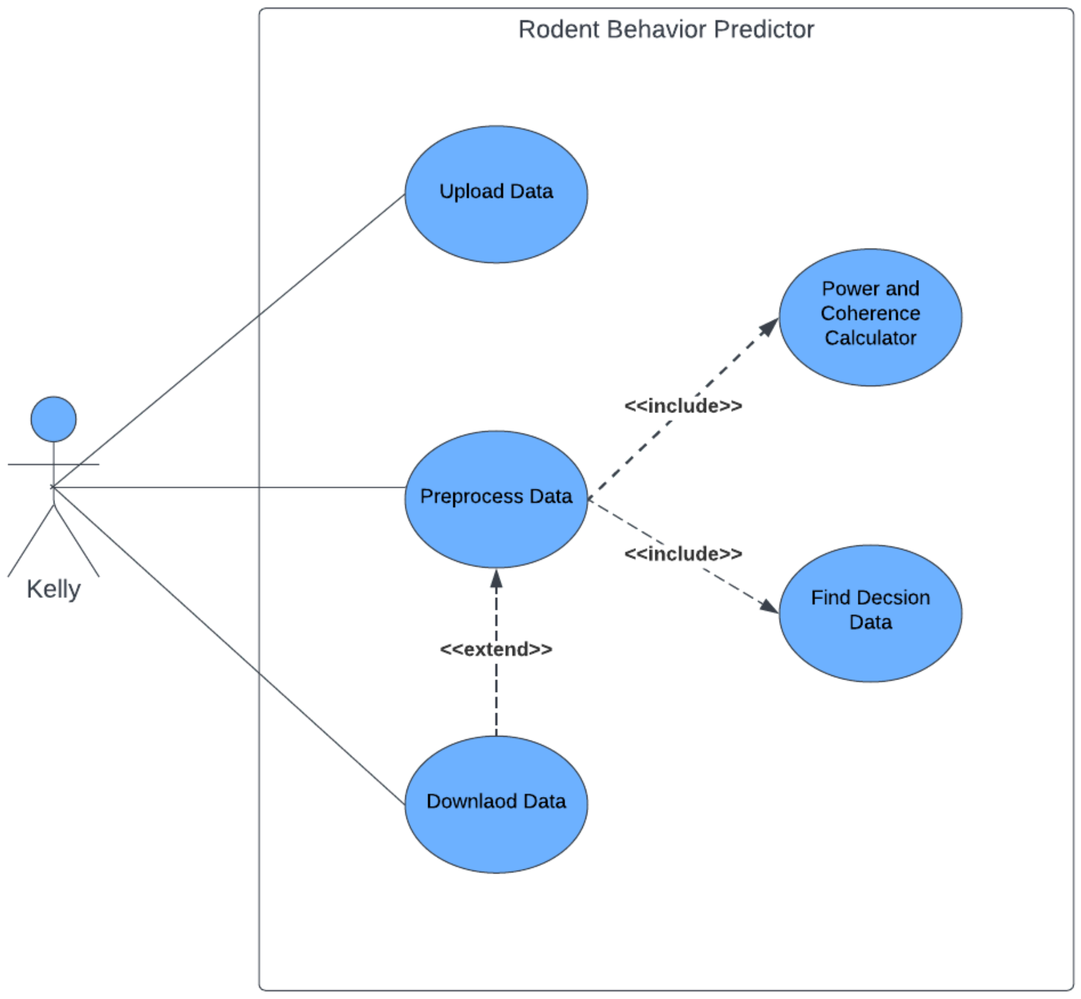
The model we build could be used by future research teams to better understand the link between neural processes in the brain to real life behavior. Furthermore, if researchers come to understand these mechanisms they could attempt to manipulate them to better help people who suffer from addictive behavior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

II. System Requirements and Specification

II.1. 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 three parts of the rodent's brain; The medial prefrontal cortex (mPFC), the nucleus accumbens shell (NAcSh), and the central nucleus of the amygdala (CeA). After collecting the data, Kelly would like to preprocess the data, so she only gets the LFP values measured when the rodent was in the decision-making process of drinking alcohol. To do this, Kelly opens our desktop application.

Once the application is opened, Kelly uploads LFP data for male alcoholic rodents into the app. Kelly then uses the program to clean the data by converting the LFP values to power and coherence values. Our machine learning model will then take these values and return the start and end timestamps of the recorded data, along with the power and coherence values derived from the LFP values recorded during this time for each rodent in the group. The timestamp outputs represent the period in which a rodent considers consuming alcohol. After preprocessing the data, Kelly saves the outputted data onto her computer and repeats the same process on the remaining three rodent groups. Kelly then closes the desktop application 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 opening our desktop application. She then loads each data file she would like the model to learn from into the application. Once the files are loaded, she can give the application her login credentials for Kamiak and command the application to ssh into the Kamiak server. The job will be sent to Kamiak, and Kamiak will build the model. The model file will be returned and can be saved onto Angela's local PC for use in the future.

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 desktop application to test the accuracy of the new model.

To start, Angela loads the new model into our desktop app. Then, she selects the option to test the accuracy. She is prompted to upload her own sample data or use one that is preinstalled into the app. 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. Regardless of the chosen option, 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.

II.2. Functional Requirements

II.2.1. [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

II.2.2 [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

II.2.3. [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 1: Desirable Function

II.2.4 [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 will be taken in 5 second chunks of time. 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: Priority Level 0: Essential and required for functionality

II.2.5 [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. The desktop application will prompt the user for their WSU credentials and use those to ssh into Kamiak and send the requested job to the cluster, with the output sent back to the desktop app.
Source: Dr. Henricks has informed us that she would like us to use the Kamiak Cluster to speed up calculations.
Priority: Priority Level 0: Essential and required for functionality

II.3. 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.

III. System Evolution

We expect system development to be a big factor in the design of this project. It's highly likely that the Henricks lab will request the aid of future capstone teams to help with their research, so we need to make sure our code is accessible to future developers. Additionally, we need to write protocols for use and ensure strict documentation so that members of the Henricks lab have many tools available to them in the case that they want to maintain or expand upon this software. We want to use Python instead of the original project language Matlab. Python is a common language that's useful for machine learning and is more widely used than Matlab. This will make the software easier for future developers to maintain.

II. System Overview

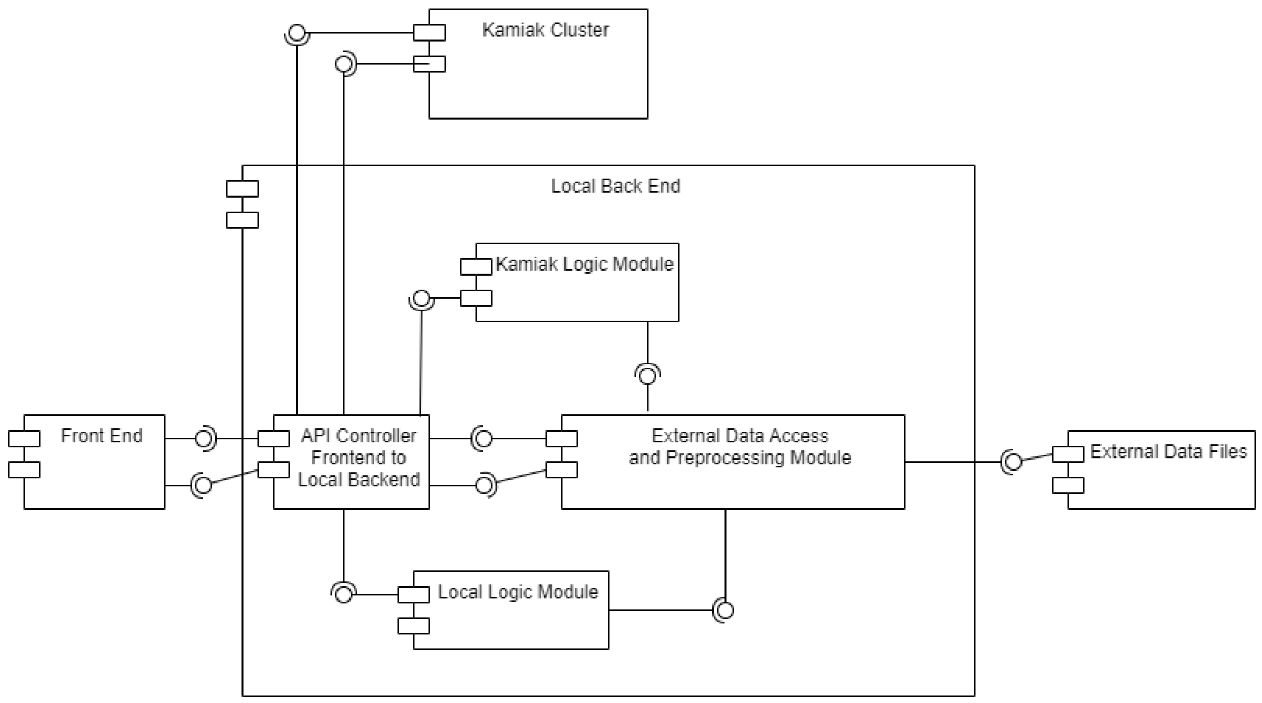
This program will perform two main tasks. The program should build the machine learning model training data provided by the user. The program should also be able to take in LFP (local field potential) data collected from the rodents brain and preprocess the data. In order to preprocess the data, it is broken up into 5 second periods. Using the data found within these 5 second periods, the cohesive and power values are calculated. After these values are calculated, we then move onto the machine learning step. Using a machine learning model the user will pass in the preprocessed data. During this step, the program should output the 5 second periods of data where the rodents were in the decision making process of drinking alcohol. The user should be able to take this outputted data and download it to their computer as a .csv file.

III. Architecture Design

III.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. The Client-Server model also more accurately reflects the relationship any local program would have with Kamiak, since we want a desktop application that sends requests to the Kamiak Cluster to run jobs and return the results to the user. We will design a frontend desktop application with a GUI for the researchers to use. Previously they have been using a terminal to use the code, so we want to make the machine learning process more accessible to the researchers. The desktop app will also have a local backend that stores the logic for the program and acts as an intermediary between the frontend and Kamiak Cluster. This local backend will also run less intensive computations that we don't need Kamiak for.

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 GUI interface where the user will input commands and view results of their requests. We have split the Server part of the Client-Server model into two parts: the Local Back End and the Kamiak Cluster. We don't want to heavily couple the GUI and logic components, so the Local Back End interprets requests from the Front End and sends job protocols to Kamiak. The Front End communicates with the Local Back End via the API Controller module. The API Controller uses the External Data Access and Preprocessing Module to preprocess data from the files inputted by the user. Then, depending on the job requested by the user, we move to either the Local Logic Module or the Kamiak Logic Module. The Local Logic Module stores algorithms for less computationally intensive processes, like testing a machine learning model against sample data or using the model to predict rodent behaviors. If we discover that this is too intensive for a local machine, we will remove this module and instead have all jobs run on Kamiak. If an intensive job is requested, like building a new machine learning model, the API will use algorithms stored in the Kamiak Logic Module and send a job request. Kamiak will run the job and return the results to the API, which will then send those results to the Front End for the user to view.



III.2. Subsystem Decomposition

I.1.1. API Controller Frontend to Local Backend

a) Description

The API Controller is the connection between our frontend GUI and the backend of our software. It handles job requests from the frontend and uses the appropriate logic module to perform those jobs, as well as returning the results of those jobs to the frontend. It also uses pre-written protocols to interface with the Kamiak Cluster when jobs that are too computationally intensive to be run locally are requested.

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 modules and preprocessing module, and one class with methods to send and retrieve output data from the kamiak system. The kamiak system will be responsible for training a predictive model. This model should be sent back to the local system where a local computation can be used to figure out the decision making periods of the rodents. Data should also be sent to the kamiak system to be used for training the model.

c) Interface Description

<u>Services Provided:</u>

1. Service Name: SendTrainingData(Data)
   Service Provided To: Kamiak Cluster

Description: Send the preprocessed data to the Kamiak cluster so Kamiak can build a model off the data.

2. Service Name: SendAlgorithm
   Service Provided To: Kamiak Cluster
   Description: Sends the machine learning algorithm code file to Kamiak to be run.

3. Service Name: SendProtocol
   Service Provided To: Kamiak Cluster
   Description: Sends a protocol to the Kamiak Cluster that tells the remote system how to run the algorithm sent to it.

4. Service Name: AccessKamiak
   Service Provided To: Kamiak Cluster
   Description: The API accesses the Kamiak Cluster via ssh with user credentials.

5. Service Name: FetchCredentials
   Service Provided To: Front End
   Description: Requests the user to enter their login credentials so the API can access Kamiak.

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

7. Service Name: ReturnOutput(Output)
   Service Provided To: Front End
   Description: Sends the timestamps of when the rodents were in the decision making process of consuming alcohol 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: SendAlgorithm
   Service Provided From: Kamiak Logic Module

4. 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 accessing user inputted data and preprocesses it for use in the logic modules. This data can be sent to either logic module to fill in the algorithm requested by the API Controller. 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.

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 data structure. We are currently unsure on what type of data structure would be appropriate to use due to the fact that we haven't received sample data from our client yet. Before calculating the power and coherence values, the subsystem should have a function that cleans any noise from 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: Local Logic Module and Kamiak Logic Module

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 that can be run locally. Since it takes time to reserve Kamiak's computing time, we intend on using the local backend to run smaller jobs that don't need Kamiak, instead of relying on Kamiak for all computations. Local computations include testing 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 move tasks in the Local Logic Module to the Kamiak Logic Module to allow Kamiak to perform those computations instead.

b) Concepts and Algorithms Generated

The Local Logic Module should have a function that runs a machine learning model sent from the API Controller on the 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

I.1.4. Kamiak Logic Module

a) Description

The Kamiak Logic Module holds algorithms for jobs that need to be run on the Kamiak Cluster. When the API Controller asks for a job to be run that needs Kamiak's computing power, it will retrieve the appropriate algorithm from the Kamiak Logic Module.

b) Concepts and Algorithms Generated

This module does not run locally; its algorithms will be too intensive with the amount of data it will be processing, so it will always be run remotely on Kamiak. It stores the algorithm for building a machine learning model, but if we find that the tasks we wish to run locally are too intensive, this module will store those as well.

c) **Interface Description**

Services Provided:

1. Service Name: BuildModel(Data)
   Service Provided To: API Controller
   Description: This service takes in training data in the form of power and coherence values and outputs a machine learning model.

Services Required:

1. Service Name: CalculatePower(Data)
   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 dictionaries and 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.

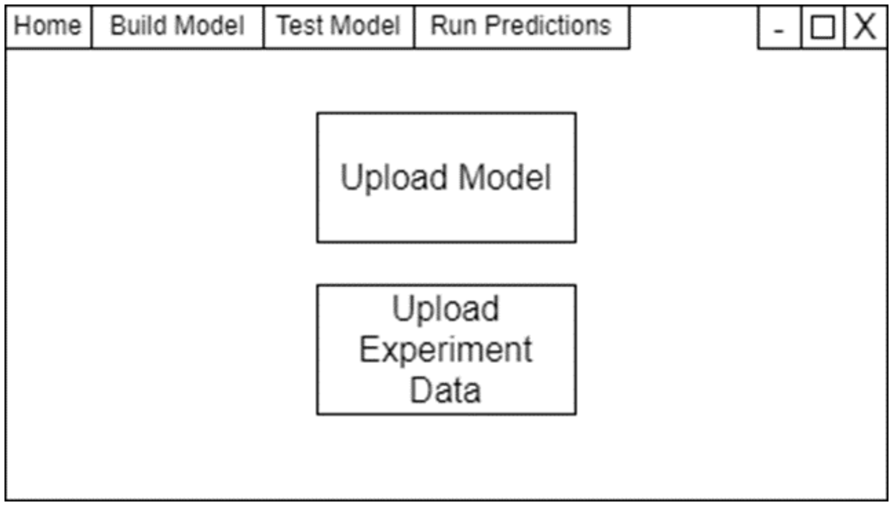
V. User Interface Design

Previously, the Henricks Lab had used a terminal to run MatLab code. This wasn't accessible to lab members who didn't have coding experience unless an in-depth protocol was written. With our switch from MatLab to Python, Nunnerson Computing intends to use Tkinter to design a General User Interface that will make the code more accessible. We expect the lab members to use this interface to more easily use machine learning to predict when their rodent test subjects are making a decision whether to self-administer alcohol. This software will be installed on one of the lab computers.

The GUI will consist of a window similar to standard desktop applications that are familiar with most PC users. Upon opening the application, the first window the user will see is the 'Home' window. The top left corner of each window will have a row of buttons, starting with the 'Home' button. The 'Home' button returns the user to the 'Home' window. Next is the 'Build Model' button. This button takes the user to a window which prompts them to upload data to train the machine learning model. Once the model is trained, the window will allow the user to download the model for future use. The next button is the 'Test Model' button. This button takes the user to a window which prompts them to upload a machine learning model. The user can upload testing data if they wish, but they can also choose to use sample data that is saved to the program. The program will test the model and return its accuracy to the window. The last button is the 'Run Predictions' button. This button takes the user to a window which prompts them to upload a machine learning model and experiment data that they would like to make predictions on. Once the program has made its predictions it will offer to the user to save the predictions to a .csv file. Each of these windows will also have a progress bar showing how much of the data has been processed.

This section mentions each use case in the Requirements Specifications document. However, the next time we meet with Dr. Henricks we will still discuss the design of the GUI with her and her team to decide whether it fits their needs.

Below is a simple example of the window that is present after a user clicks the 'Run Predictions' button. We intend on keeping the GUI simplistic in its design unless our client desires more features, like displays for data.



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.

VII. 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.

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.

SURCA (Showcase for Undergraduate Research and Creative Acts): A WSU event showcasing undergraduate research.

VI. 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]