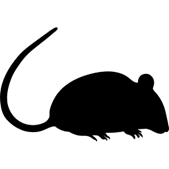
Machine Learning Approach to Identifying Neural Features That Predict Rodent Behavior

**Solution Approach**

Washington State University - Department of Psychology



Nunnerson Computing



Aidan Nunn, Charlie Nickerson

10/5/22

**TABLE OF CONTENTS**

**I. Introduction**

In the document, the Nunnerson Computing team will present the solution approach for building a program that can predict when rodents addicted to alcohol are in the decision making process of consuming alcohol using the lasso machine learning model. The purpose of writing this is to help future researchers understand the basic architecture and structure of our code.

We will first go over the architecture and design of the program and give a top-level design view of the software architecture and describe each component and its responsibility. In the next section we’ll provide the subsystems of our program and describe the rationale for the decomposition of each subsystem. Next we will talk about the type of data structures and databases required for our program. After that we will end the paper by describing the user interface design.

**II. System Overview**

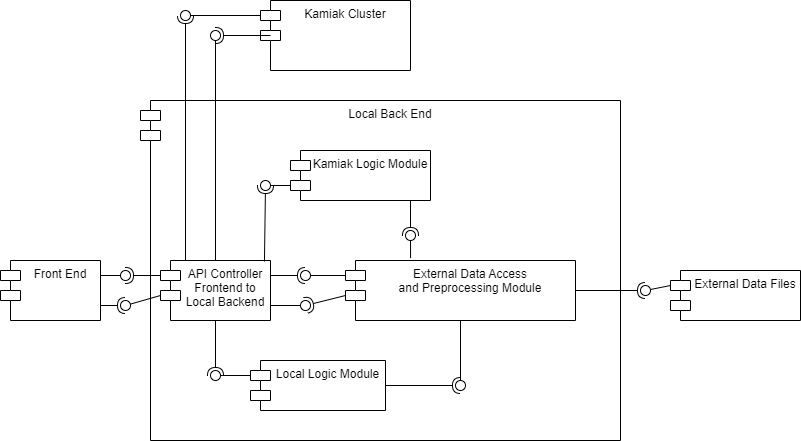
This program will take in data which contains the LFP (local field potential) data collected from the rodents brain and preprocess the data. In order to preprocess the data, it is broken up into 5 second periods. Using the data found within these 5 second periods, the cohesive and power values are calculated. 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 Frontend to Local Backend module (API F/B). The API F/B branches to the External Data Access Object Interface, which preprocesses data from the files inputted by the user. Then, depending on the job requested by the user, we move to either the Logic Module Local or the Logic Module Kamiak. The Logic Module Local 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, Logic Module Kamiak uses the API Local Backend to Kamiak module to interface with Kamiak and send a job request. Kamiak will run the job and return the results.



**III.2. Subsystem Decomposition**

**I.1.1. API Frontend to Local Backend**

1. **Description**

The API Frontend to Local Backend (API F/B) 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.

1. **Concepts and Algorithms Generated**

This subsystem will contain one class with methods to send data 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.

1. **Interface Description**

Services Provided:

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.

1. Service Name: DisplayOutput(Output)

Service Provided To: Front End

Description: Displays the timestamps of when the rodents were in the decision making process of consuming alcohol to the screen.

Services Required:

**I.1.2. External Data Access Object Interface**

1. **Description**

The External Data Access Object Interface preprocesses data from inputted user data files for use in the logic modules.

1. **Concepts and Algorithms Generated**

The External Data Access Object Interface 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.

1. **Interface Description**

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.

Services Provided:

1. Service Name: LoadData(File)

Service Provided To: API Controller Front End to Back End

Description: Read in the external data files and puts it into a data structure

1. Service Name: WriteToFile(Data)

Service Provided To: External Data Files

Description: Writes the output data neatly to a CSV file

1. Service Name: CalculatePower(Data)

Service Provided To: API Controller Front End to Back End

Description: Takes in a list of LFP values and returns a list of power values.

1. Service Name: CalculateCoherence(Data)

Service Provided To: API Controller Front End to Back End

Description: Takes in a list of LFP values and returns a list of coherence values

1. Service Name: CleanNoise(Data)

Service Provided To: API Controller Front End to Back End

Description: Takes in a list of LFP data and returns a noiseless list of LFP values.

Services Required:

**I.1.3. Local Logic Module**

1. **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 remove the Logic Module Local and allow Kamiak to perform all computations instead.

1. **Concepts and Algorithms Generated**

The local logic module should have a function that runs a 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.

1. **Interface Description**

Services Provided:

1. Service Name: FindTimestamps()

Service Provided To: Front End

Description: Runs the predictive model on preprocessed data and returns timestamps of when the rodents are consuming alcohol.

Services Required:

**I.1.4. Kamiak Logic Module**

1. **Description**

The Logic Module Kamiak holds protocols and algorithms for jobs that need to be run on the Kamiak Cluster. When the API Frontend to Local Backend asks for a job to be run that needs Kamiak’s computing power, the Logic Module Kamiak will send the appropriate algorithm to the API Local Backend to Kamiak.

1. **Concepts and Algorithms Generated**

This module should have a function that takes in code with data that was sent from the API controller to build a machine learning model. This system should also have a function that sends the model back to the API controller where it will be sent to the local logic module.

1. **Interface Description**

Services Provided:

Service Name: BuildModel(Data)

Service Provided To: API Controller Front End to Back End

Services Required:

**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.

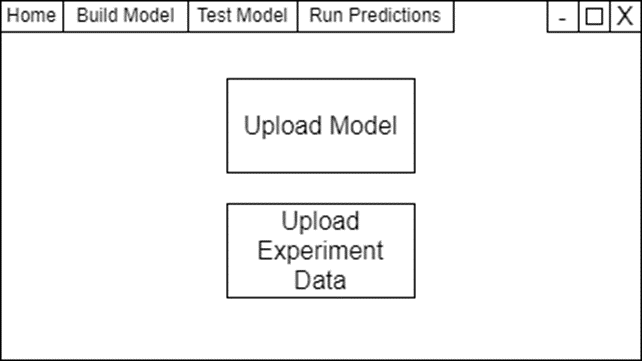
**IV. User Interface Design**

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 .txt 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 an example of the window that is present after a user clicks the ‘Run Predictions’ button.



**IV. Glossary**