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.

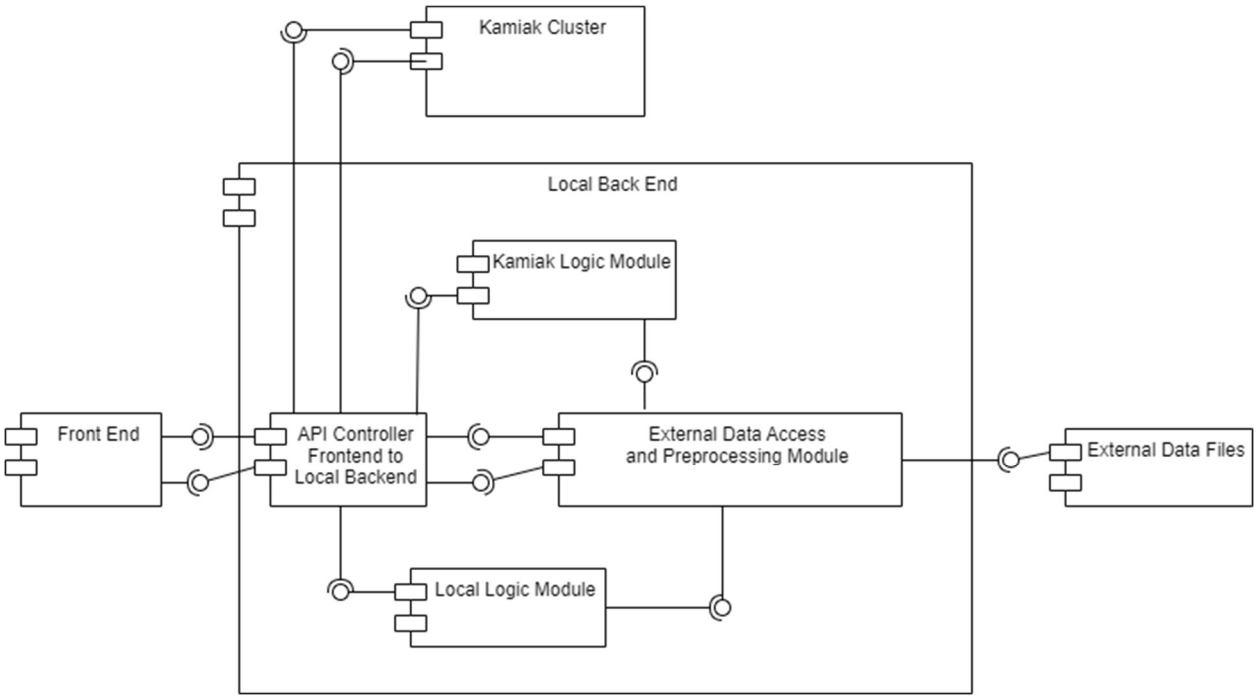
III. Architecture Design

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 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

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.

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

Solution Approach 5

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

<u>Services Provided:</u>

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 arrays to store LFP data will be sufficient. An array will also be used to store the sample testing data.

IV. User Interface Design

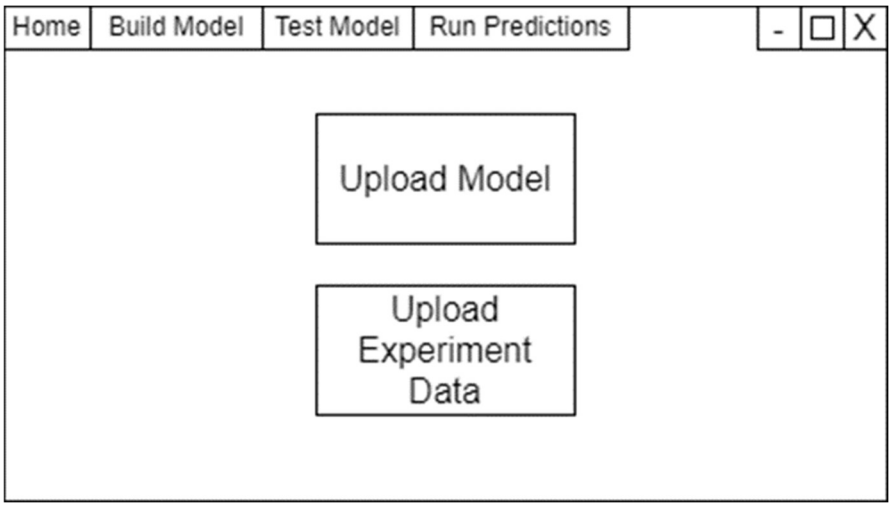
Previously, the Henricks Lab had used a terminal to run MatLab code. This wasn't accessible to lab members who didn't have coding experience unless an in-depth protocol was written. With our switch from MatLab to Python, Nunnerson Computing intends to use Tkinter to design a General User Interface that 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 .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.



IV. 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..

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).

Frontend: Client-side of an application. The part the 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.
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.

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.

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