Engineering a Web-based Interface to Predict the Unknown Physical Properties of Solar-like Stars

Alejandra Perea Rojas[1,⋆]
Supervised by Earl P. Bellinger[2]

[1] Fox Lane High School, e-mail: aperearojas@student.bcsdny.org
[2] Stellar Astrophysics Centre, Aarhus University, Denmark, e-mail: bellinger@phys.au.dk

ABSTRACT

We present Starithm, a web-based interface with the ability to predict the physical properties of solar-like stars from observations with the use of machine learning. The architecture of Starithm consists of two separate frameworks: a back-end model and a dynamic front-end web application. In this paper, I focus on the development, optimization, and linkage of these frameworks. Furthermore, I present the engineering behind the user database and machine learning interface. I discuss the implementation of a database-backed user structure, asynchronous front-end execution, email delivery server, and online database for storage of data and results.

Keywords. Stellar astrophysics, machine learning, full-stack development, asynchronization.

1. INTRODUCTION

Stellar pulsations provide great insight into a star's unknown physical parameters. Asteroseismology—the study of these internal waves in stars—can be used to determine properties of the stellar core and measure the mass, age, and chemistry of a star [1]. Thanks to missions like the NASA *Kepler* space observatory, there is now asteroseismic data available for more than 100 solar-type stars [2].

Using computational methods such as iterative optimization, genetic algorithms, Markov-Chain Monte Carlo, asteroseismology has been used to determine the unobservable parameters such as stellar age with an estimated relative error of less than 10%. However, these methods can be very slow, taking sometimes days or weeks to arrive at an answer. To remedy this situation, Bellinger et al. [1] introduced a quick machine learning algorithm that is able to predict unknown stellar parameters with solar-like oscillations in less than one minute.

In this paper, I present a way to further this tool into an even more accessible device: to connect the machine learning algorithm to a web-based application. The design of the application allows improved performance and provides additional features (such as visuals and other parameters). The application also has a user-backed database, an interface delivering results with an email-server, and storage capability for stellar information in an online public catalog.

1.1 Architecture Goals. An interface is the shared sector between two or more computer components that would not otherwise connect. In terms of web development, an interface can transform a static website into a dynamic page with the implementation of a back-end algorithm. To engineer a web-based interface, the front-end and back-end frameworks discussed in Section 2 must be linked through an external server. I have engineered the interface of Starithm with PHP (internally) and Python (externally). This will be discussed in Section 3. The application structure is visualized in Figure 1.

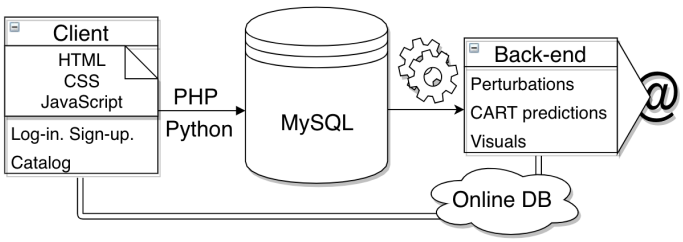

Fig. 1: An overview of the application's overall structure and design.

2. ARCHITECTURE

The Starithm web-based application consists of two separate frameworks adjoined by an interface—the back-bone of the application. In this section, I will provide an overview of the core design and architecture of the application's primary building blocks.

2.1 User Database. To transform a static web-application design into an interactive one, a user-based database was implemented to the front-end web application. Facilitating the work-flow, the user-based database allows for the separation of back-end requests and results. It allows for greater organization functional for a substantial amount of data. This data can then be automatically re-arranged to be placed in an online database available for non-users.

2.2 Back-end Machine Learning Algorithm. The back-end application takes in measured stellar pulsation frequencies and other observations, such as the effective temperature of the star from spectroscopic measurements. Prior to training the machine learning algorithm, it perturbs the obser-

⋆This document was typeset using LaTeX by the author. All figures were created by the author.

vations to account for measurement uncertainty. Afterwards, it trains a Classification and Regression Trees (CART) algorithm on 350,000 theoretical stellar models to learn the matching between observable and unobservable aspects of the star. It then uses this trained regression model to predict the ages and other properties of the star from its observations.

Through automatic file relocation from the web-based interface, the machine learning algorithm is able to access and use the individual stellar inputs belonging to each user. To increase the execution speed, a *joblib* module (Python) allows for the implementation of the saved training for the random forests specific to a certain type of parameters, preventing consecutive repetitive random forest training.

3. INTERFACE

The building blocks of Starithm are the back-end and front-end frameworks described in Section 2. In this section, the components of the interface will be described in greater detail.

3.1 Front-end and User Database. I initially developed a static website, and further, I implemented a user-backed database using MySQL to optimize the client executions and data organization. The database type I chose was InnoDB, a newer version unlike MyISAM, because of faster tables for new input with row-locking. I built a *session_start()* PHP method to send information back and forth the user database and the front-end web application. With a running session, the connection is achieved with the input of host, port, user, and password. From there, a series of forms and requests were developed.

3.2 User Interface. I continued the linkage of the front-end and user database with PHP POST methodology, allowing for user registration and the connection to clients. To create a user, a request is delivered (using PHP POST) with the following parameters: first name, last name, user-name, email, and password. Password safety is provided with the *MD5* message-digest algorithm. Once an account is created, through a different PHP POST request, the database running session is transferred to a personalized client web-page. This page, through the interface, sets up the user's individual directory for the back-end programs and data.

3.3 Data and Security. To facilitate security of input information, the prevention for MySQL injection is assured via query sanitation, and restrictions to the type of files uploaded are placed on the file name, size, and type. After the client files are uploaded, the user has an option for asynchronous execution. File and data relocation were done to follow user database structure.

3.4 Asynchronization and Optimization. Parallel or asynchronous programming allows for the simultaneous computation of the different components of an algorithm. I built an asynchronous execution into the interface to have the back-end structure execute independently of the client, preventing web-delay in the execution. This was accomplished with the *subprocess* module in Python. To optimize the execution, the *joblib* and *pickle* modules were implemented to make further predictions more efficient with previously saved training of random forests. The storage of saved training is 8 GB on average.

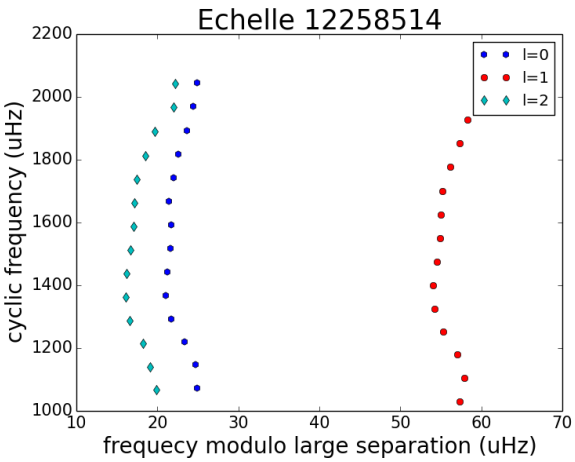

Fig. 2: Echelle diagram showing the observed oscillations for the solar-like star KIC 12258514. The star has a large frequency separation of 74.79 μHz (see text). Each point is an individual mode of pulsation observed in this star. The colors represent the spherical degrees (l) of the oscillation modes.

3.5 Email-Server. I implemented to the interface an email-server to compile and deliver the results to the client. The *email* package from Python was used with the MIME modules and encoders, and the Simple Mail Transform Protocol (SMTP) was used as a server for a Gmail website account. I implemented another email-server for password recovery through PHPMailer and SMTP as well.

3.6 Online Catalog. After a client has submitted observations for a stellar object, the interface will automatically store the individual results for the object in a separate directory in the catalog. I created this in the front-end side for public accessibility, and I implemented a search engine with a PHP POST matching method and a JSON tagging and encoding function.

3.7 Echelle. The frequencies of radial oscillations in pulsating stars are roughly evenly spaced by a quantity known as the large frequency separation, $\Delta\nu$. Echelle diagrams consist of the segmentation and stacking of an oscillating star's pulsation frequencies based on its measured $\Delta\nu$. An Echelle diagram for an example star is shown in Figure 2. Because of the clear change of variations in the diagram, Echelle plotting is useful for the visualization of the stellar oscillations, and also for stellar parameter determination. For example, comparing Echelle diagrams of two stars can be used to estimate the otherwise unknown ratio of their mean densities [4]. For more information on Echelle diagrams, I refer the reader to Bedding et al. (2010) [4].

To incorporate this addition to the back-end program, I built an Echelle plotting function into the user interface. This automatically separates each mode frequency into its respective mode l for required mathematical manipulation. This is done for each client-uploaded stellar object.

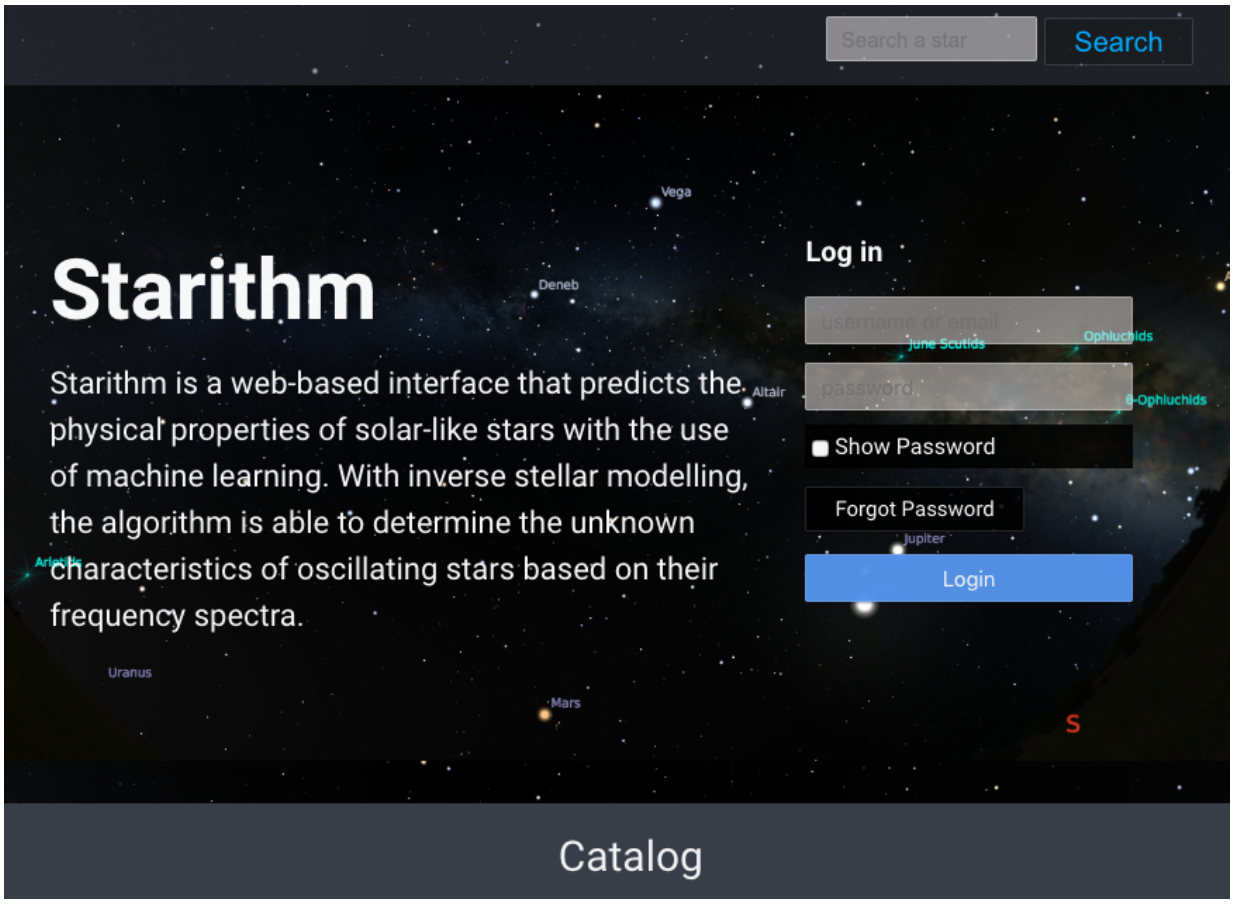

Fig. 3: The home-page of the web-application is depicted with a log-in, search engine, and catalog on display.

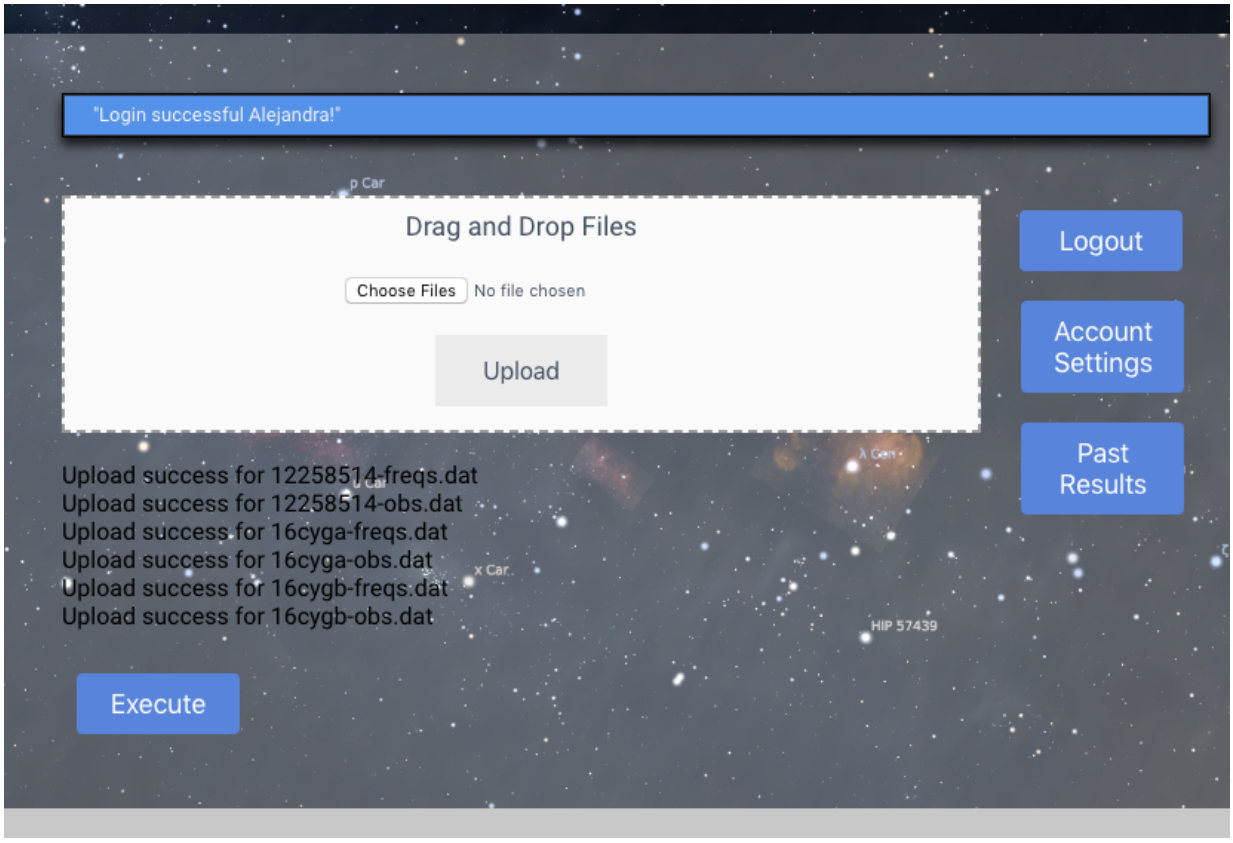

Fig. 4: The client window consists of conventional front-end user components, drag-over and selecting uploading methods, and a record of results from previous submissions (aside from the online database catalog).

Name	Mass	Y	Z	$\alpha_{\rm MLT}$	Overshoot	Diffusion
12258514	1.27 ± 0.039	0.2624 ± 0.0068	0.0223 ± 0.0029	1.83 ± 0.11	0.099 ± 0.017	2.24 ± 1.84
16 Cyg A	1.09 ± 0.022	0.2636 ± 0.007	0.0236 ± 0.0016	1.834 ± 0.08	0.072 ± 0.042	1.06 ± 0.81
16 Cyg B	1.031 ± 0.016	0.2661 ± 0.0058	0.0215 ± 0.0014	1.799 ± 0.063	0.098 ± 0.033	1.17 ± 0.59

Name	Age	X_c	$\log g$	Luminosity	Radius	$Y_{\rm surf}$
12258514	4.24 ± 0.34	0.069 ± 0.013	4.1233 ± 0.0045	3.06 ± 0.16	1.62 ± 0.02	0.222 ± 0.028
16 Cyg A	6.86 ± 0.71	0.072 ± 0.032	4.2954 ± 0.0066	1.568 ± 0.084	1.231 ± 0.014	0.2444 ± 0.0086
16 Cyg B	6.89 ± 0.42	0.149 ± 0.033	4.3548 ± 0.0076	1.243 ± 0.034	1.118 ± 0.013	0.245 ± 0.011

Table 1: Short segment of the email output sent to the client. This segment shows the predicted average parameters and their uncertainties for the solar-like stars 16 Cygni A & B and KIC 12258514. Ages are given in billions of years; mass, radius and luminosity are in solar units; initial helium abundance Y, surface helium abundance $Y_{\rm surf}$, core hydrogen abundance X_c, and metallicity Z are given in fractional units; and $\alpha_{\rm MLT}$, overshoot, and diffusion refer to theoretical parameters of stellar structure.

4. RESULTS AND CONCLUSION

The general structure of Starithm follows with the Architectural Goals described in Section 1.1. The front-end application was structured off a static web-design consisting of HTML and CSS, and it connects the client to the user-based database and back-end algorithm through an interface which was built with PHP and Python. The homepage of the website is displayed in Figure 3.

Much of the methodology from the internal client side is achieved through PHP POST methods that link the website to the user-backed database supported by MySQL through a PHP connection and online session. New users are stored in the database, and the internal interface handles the automatic creation with a necessary set-up of the location for the incoming client data. Restrictive parameters and sanitation were implemented for the uploaded data and client input respectively, but further procedures for extended security will be required prior to deployment. An example of a client-view for an existing user is shown in Figure 4.

To execute the back-end machine learning algorithm, an execution with a PHP POST method will request the external interface built with Python to asynchronously run the perturbations and CART algorithms. To test the application, the frequency spectra and initially observed parameters of three solar-like oscillating stars were submitted through the user shown in Figure 4, and the application successfully predicted and delivered the expected output to the client's email. A segment of this output is demonstrated in Table 1.

Another implementation with the *pickle* module from Python was implemented but could not store random forests. As a result, this method was switched with a *joblib* module which successfully stored random forests but crashed in their loading. Furthermore, there needs to be an evaluation on how either the pickle or joblib Python modules will handle different input parameters. The initial asynchronous execution runs smoothly and prevents client-side crashing, but there needs to be further work on the back-end side of the application.

Nevertheless, the full-stack web application, in its entirety, is successful (with architectural and accuracy goals). The web application successfully handles incoming and existing users and data. It properly executes the CART algorithm and delivers the output which is then stored. All components of the interface fulfill their respective tasks, and the predictions are made accurately. The novel engineering detailed in this paper is another further advancement and potential successful tool for stellar research. Furthermore, we plan to deploy it simultaneously with the submission of a future publication in which we also extend the back-end algorithm to handle more kinds of stars.

Acknowledgements. I would like to thank Dr. Bellinger for this wonderful scientific opportunity that would not have been possible without his guidance. I would also like to thank my school's Science Research teacher, Ed.D. Candidate, Stephanie Peborde-Burke, for her amazing teaching, and I want to thank my parents for their infinite support and hard-work.

Software. The programming languages used for the engineering of this website are the following: for front-end web development, HTML, CSS, and JavaScript; for interface programming, Python and PHP; and for the back-end component, MySQL, PHP, Python, and R.

5. REFERENCES

[1] Bellinger, E. P., Angelou, G. C., Hekker, S., Basu, S., Ball, W. H., & Guggenberger, E. (2016). Fundamental Parameters of Main-Sequence Stars in an Instant with Machine Learning. The Astrophysical Journal, 830(1), 31.

[2] Bellinger, E. P., Angelou, G. C., Hekker, S., Basu, S., Ball, W. H., & Guggenberger, E. (2017). Stellar Parameters in an Instant with Machine Learning-Application to Kepler LEGACY Targets. In EPJ Web of Conferences (Vol. 160, p. 05003). EDP Sciences.

[3] Di Mauro, M. P. (2017). A review on Asteroseismology. Frontier Research in Astrophysics II.

[4] Bedding, T. R., & Kjeldsen, H. (2010). Scaled oscillation frequencies and échelle diagrams as a tool for comparative asteroseismology. arXiv preprint arXiv:1001.5038.