



LabRat - A Machine Learning Platform for Collaborative Optimized Predictions

Julio M. Fernández Jiménez
Bioinformatics and Statistics
Software Development

Elisabeth Ortega Carrasco
Jose Antonio Morán Moreno

06/2018



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada [3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada [3.0 España de Creative Commons](#)



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-CompartirIgual [3.0 España de Creative Commons](#)



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial [3.0 España de Creative Commons](#)



Esta obra está sujeta a una licencia de Reconocimiento-SinObraDerivada [3.0 España de Creative Commons](#)



Esta obra está sujeta a una licencia de Reconocimiento-CompartirIgual [3.0 España de Creative Commons](#)



Esta obra está sujeta a una licencia de Reconocimiento [3.0 España de Creative Commons](#)

B) GNU Free Documentation License (GNU FDL)

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

C) Copyright

© (Julio Manuel Fernández Jiménez)

Reservados todos los derechos. Está prohibido la reproducción total o parcial de esta obra por cualquier medio o procedimiento, comprendidos la impresión, la reprografía, el microfilme, el tratamiento informático o cualquier otro sistema, así como la distribución de ejemplares mediante alquiler y préstamo, sin la autorización escrita del autor o de los límites que autorice la Ley de Propiedad Intelectual.

Project Technical Card

Project Title:	<i>LabRat - A Machine Learning Platform for Collaborative Optimized Predictions</i>
Author name	<i>Julio M. Fernández Jiménez</i>
Consultant name:	<i>Elisabeth Ortega Carrasco</i>
Professor name:	<i>Jose Antonio Morán Moreno</i>
Submission date (mm/yyyy)	<i>06/2018</i>
Title:	<i>Bioinformatics and Statistics Master's Degree</i>
Project area:	<i>Software Development</i>
Project language:	<i>English</i>
Keywords	<i>Machine learning, R, privacy, optimization, data modeling, prediction, SaaS, API, Rest</i>
Project summary (250 words max.): <i>Should include goal, aplicaron contexts, methodology, results and work conclusions.</i>	
<p>LabRat is a SaaS designed for machine learning analysis capable of selecting the best algorithm with the right parameters for a given data set. The system is capable of automatic and manual model training and prediction. It intends to automatize the machine learning training and prediction process by providing an easy to use web based UI application.</p> <p>LabRat is built under the PHP Laravel framework, which provides a UI web based application and a secure API Rest infrastructure. The R data analysis engine is built under a Node.js API Rest server that communicates with the Laravel component as a micro-service. Data is sent to the R engine via SFTP.</p> <p>The software development process leads to an inexpensive open-source and private alternative to other commercial machine learning tools in the market. The software can be downloaded and installed into any local machine or server providing privacy and as much computer power as desired. The system micro-service infrastructure allows the user to scale each of its service component as needed.</p> <p>This platform is able to automatized the bioinformatician work when it comes to machine learning data training and prediction analysis. It provides a single solution to a large variety of unsolved AI problems with the possibility of scaling to a more robust product that can adapt to a larger and wider set of machine learning problems.</p>	
Abstract (250 words or less):	

Motivation

To provide the scientific community with an open-source, inexpensive, scalable and automated machine learning platform alternative similar to other commercial services in the market and mostly designed for small to medium size research teams.

Objective

To develop LabRat, a micro-service oriented machine learning web and API Rest platform for data modeling and prediction based on R.

Methodology

The system was developed in Laravel, a Symfony based framework for PHP, Node.js and R. The Laravel main back-office platform is a web based UI that allows the user to interact with the R engine. Data is sent to the R server for modeling or prediction via SFTP, where the R server uses it to train models and generate predictions on demand. The R server is manipulated via API Rest through the web based back office.

Results

Although LabRat can be considered to be an ongoing work in progress, a web based back office and an R engine server were developed and offered as an open-source solution to machine learning analysis at Git-Hub. The final product offers the necessary set of tools for machine learning model training and prediction and serves as a starting point for a more robust software development.

Table of Content

Project Technical Card	i
Table of Content	iii
Figure List	4
1. Introduction	1
1.1 Context and project justification	1
1.2 Project goals	3
1.3 Project procedure and description	3
1.4 Project planning	6
1.5 Deliverable brief description and summary	10
1.6 Remaining chapters and content brief description	10
2. Technical Design	11
Project structure	11
Development process	11
Functional requirements	12
Non-functional requirements	12
UML model design	14
Software installation requirements	19
3. Risk Analysis and Development Strategies	20
Initial development risk analysis	20
Development strategies and troubleshooting	20
4. User Manual	24
Demo datasets	24
Project security	24
System Workflow	26
Model training	27
Data prediction	29
User graphical interface	29
Economical Evaluation	30
Production costs	30
Annual fixed costs	30
Product Viability	30
5. Conclusions	32
Goals Analysis	33
Methodology and Planning Challenges	34
Improvements and future software development guidelines	34
6. Glossary	37
7. Bibliography	40

Figure List

TABLE 1: PROJECT TASK LIST
IMAGE 1: PROJECT GANTT DIAGRAM.
IMAGE 2: UML MODEL DESIGN
IMAGE 3: MICRO-SERVICE INFRASTRUCTURE
IMAGE 4: COLLABORATOR FORM
IMAGE 5: APPLICATION DATA WORKFLOW
IMAGE 6: MODEL TRAINING FORM
IMAGE 7: MODEL STATE LIST
IMAGE 8: MODEL TRAINING FORM
IMAGE 9: DATA PREDICTION FORM
IMAGE 10: PROJECT LIST
TABLE 2: PRODUCTION COSTS
TABLE 3: ANNUAL FIXED COSTS

1. Introduction

1.1 Context and project justification

The motivation behind LabRat is to democratize machine learning techniques among scientific community members by providing a code-free and easy-to-use machine learning platform. It allows scientists to create models from raw data without the use of any scripting language such as R or Python.

According to my personal experience throughout the Bioinformatic's Master's Degree studies, the use of machine learning as a tool for diagnosis and data exploration is limited to the programatic design of custom made algorithms and its implementation to a specific science problem.

This procedure requires of great knowledge in the use of third-party libraries for either Python or R, not to mention the data pre-processing and algorithm adjustment process needed to come up with the best predictive model for a specific problem. As a matter of fact, many bioinformatic research papers solely focus on the intent to identify the best machine learning algorithm and the optimum parameters to improve a certain problem prediction outcome.

Another important backdraft when working with machine learning analysis is the computer processing power and its capacity to analyze data. This is more noticeable when working with multilayer neural networks.

Although services such as AWS Machine Learning servers provide an efficient and comfortable solution for the hardware and machine power problem, their services are expensive and never completely private: "An in-the-cloud service will always be someone else's computer."¹

Data and model managing can also be a hazard when working with tools such as R programing languages. If the data scope changes, the algorithm implementation may change as well. This requires for the bioinformatician to review and modify the R script in order to adapt to the new data requirements. There is a generalize lack of automatization and management tools when it comes to R scripts.

LabRat intends to be the missing gap between research data analysis and optimized modeling results by providing the researcher with a high level of privacy and customizable machine power.

¹ As cited by Elisabeth Ortega Carrasco

As the use of machine learning processes becomes widely used in bioinformatics, the need for smarter, automated and customizable tools is becoming a necessity rather than a luxury or a software commodity.

LabRat is a SaaS for machine learning model training and evaluation on demand. It is the answer to a private, cheap open source alternative to machine learning analysis.

The software developed within the scope of this project is a machine learning platform for predictive strategy with optimized multi-analysis. In other words, it is a SaaS designed for machine learning analysis capable of selecting the best machine learning algorithm with the right parameters for the given data set. In addition, all predictive models generated by LabRat are adjusted when new data is fed by either switching to a new algorithm or by modifying the current algorithm parameters.

Although automatization is the key element of LabRat, the system also allows for custom defined processes, all of this from a usable and user oriented UI. This smart predictive strategy of LabRat is indeed a solution to the tedious algorithm analysis alternative offered by other third-party libraries such as R and Python.

LabRat also offers an inexpensive and more private alternative to other commercial machine learning tools out in the market. The software can be downloaded and installed into any local machine or server providing privacy and as much computer power as desired. Since LabRat will work under a micro-service infrastructure, it will be possible to assign the available machine resources according to each service CPU requirements.

LabRat uses R for data analysis, more specifically its Caret package. The use of external, widely used and tested libraries for its machine learning analysis provides the system with a high level of trust and scalability.

As any other SaaS, LabRat comes with a multiuser UI platform for managing projects, models and users. This platform is mobile-first and come with an additional isolated API REST engine.

The concept behind the branding of LabRat relates to the use of live animals through the history of science. The LabRat branding is a tribute to the millions of animal that are sacrificed in the name of science and progress. It represents the hope and the desire of a more ethical world where the discriminated use of animal lives will no longer be linked to science and progress. A future where animals are entirely replaced by machines in the name of science is possible.

1.2 Project goals

There is only one main goal for this project that encloses the whole development process:

(1) To provide the scientific community with a collaborative open source application to generate machine learning trained models and predictions from unclassified data.

This main goal can be broken apart into a series of subtasks as described below:

(1.1) To minimized custom made scripting tasks for machine learning analysis.

(1.2) To provide a machine learning trained model environment where the scientific community can share and use multidisciplinary trained models on demand.

(1.3) To provide a private and encapsulated environment for machine learning based data analysis and storage.

1.3 Project procedure and description

It was clear from the very beginning that LabRat ought to be designed as a three layer infrastructure software, these being:

- ❖ **Front-end UI layer:** this is the front-end user-interface layer built mostly in HTML/CSS/Javascript. It is built under a non-intrusive and mobile-first technology such as Bootstrap framework and Vue.
- ❖ **Back-Office API Rest layer:** this is where the business logic resides. This is what we will call the back-office application. It comes in two modalities. The first one is an MVC platform that interacts with a well decoupled but embedded front-end layer. The other is an API Rest entry point designed for secure third party implementations. Both presentation layers work under the same service layer.
- ❖ **R-Caret machine learning scripting layer:** this is the core of the application. It contains all the machine learning scripts that will process the raw data for model training and prediction.

During the work planning brainstorm process, several approaches were evaluated before a solution came up for a definitive software architecture. Some of the proposed architectures are described below:

Node.js + AngularJS + Python

This was the initial and most exciting approach for the application software infrastructure. Whilst node and Angular are both javascript

based languages, Node (as a back-end technology) offers great framework options for API Rest approaches such as Meteor and LoopBack.

Out of these two, LoopBack was by far the most interesting approach. Unfortunately this option was discarded due to the lack of maturity of the LoopBack framework, which is currently being updated into a brand new and exciting 4.0 version.

Out of the two remaining tech choices involved in this option, AngularJS was the most solid alternative so far. It was clear from the very beginning that LabRat had to be totally decoupled when it comes to back/front end design options. On the other hand, Python was shortly discarded as a machine learning scripting language. We found the R-Caret package a very completed and widely used solution for machine learning analysis.

Sprint Boot JavaEE + AngularJS + R-Caret

This is the enterprise software architecture choice for LabRat. Although JavaEE and Spring Boot will provide the platform with a robust and scalable professional architecture, certain parts of the Spring Boot framework, such as Hibernate for ORM and DAO, lack of malleability and practicality when it comes to designing complex data models. In addition, hosting a JavaEE application could increase hosting costs dramatically.

The R-Caret package is still the best choices for machine learning scripting.

Laravel + Blade/Vue + R-Caret

Laravel is a PHP web oriented developing framework based on Symfony. It comes with very useful features already integrated such as ORM, event-driven programming complements and unit testing among others. Because it is built on PHP, Laravel is easy and inexpensive when deploying in a production environment. It is also easy to maintain and scale.

Blade is the custom front-end scripting framework or Laravel, it integrates great with the back-end code and needs no preambulars. Laravel comes with Vue or React as default JavaScript engine. Out of the two Vue was found to be more straight forward and simpler to use. Nevertheless, and in order to maintain a mobile-first non-intrusive technology approach, this implementation would come with an alternative API Rest back-end that will allow for the system to be used as a third party tool.

The R-Caret package is still the best choices for machine learning scripting and front-end.

Out of the three evaluated options, the Laravel+Blade+R-Caret package was considered to be the best option for development.

Continuous development and integration (CD/CI)

In order to provide a smooth development process and to ensure the quality of the work, an effective CD/CI process was designed to optimize the project development process. The design process is composed by the following tools:

- ❖ **GitHub**: this is the code repository chosen for the application development code control. GitHub was chosen over BitBucket because of its great integration into other CD/CI systems such as CircleCI and Heroku for the most.
- ❖ **CircleCI**: web based continuous integration platform used for code unit and integration testing. It integrates with GitHub through web hooks to block the code merging process if tests fail.
- ❖ **Heroku**: Heroku is a continuous development platform that integrates with GitHub version control repositories and perform automatic deployments in different stages of a given pipeline. Each pipeline is composed of a branch, development, staging and production serve container:
 - ❖ *Branch server app (also called “review apps”)*: is connected to Github and creates a new server instance for each active branch in the repository. When a new commit is detected for each branch, Heroku pulls the branch code and deploys the application for live review.
 - ❖ *Development server app*: this is where the develop branch is deployed for review. It auto-deploys when a new merge or commit is detected in the branch and allows the developer to preview the changes from the merged branch into the main code under controlled conditions.
 - ❖ *Staging app*: this is a pre-production stage and only deploys when the master branch is modified. It helps to review the master code in a production-like environment.
 - ❖ *Production app*: unlike the previous pipeline stages described above, the production app must be deployed manually from the staging app. This branch does not goes through the regular deployment process since it always receives a fully functional and deployed code directly from the staging app. This special deployment procedure prevents downtimes in the production server as the code migration process is performed in a matter of milliseconds and it is unappreciable by the end user.

Each of different parts that compose the project has its own Git repository, CircleCI instance (when available) and Heroku pipeline.

In order to maintain a quality control standard, a TDD oriented programming methodology was followed throughout the developing process.

Although Heroku comes with all the necessary infrastructure to get the application up and ready, Digital Ocean was chosen as a production environment for the following reasons:

- ❖ **Data persistency:** Heroku is great but it does lack of data persistency when it comes to storing files. Raw data files and model files could not be stored in Heroku and needed of a third party service, such as AWS buckets, to store data.
- ❖ **Micro-service communication:** The R engine server needs of two entry points or ports in order to function; one for the embed SFTP server and another for the R scripting remote execution. Heroku only allows for one single port to be exposed to the internet and it lacks of the necessary tools to build an intranet infrastructure.

Since one of the goals of LabRat was to build a system that could be used by any team in any computer, it made sense to use a hosted virtual machine to host the application. It was also a closer approach to reality, since research teams will use private or remote virtual machines to run the code. Digital Ocean offered low-cost virtual machines and ready for production deployment.

1.4 Project planning

The project planning and execution was carried out in Jira under the Agile methodology using the Scrum system. Each sprint was composed of two full weeks, where each day was expected to have at least four hours of work.

According to the planning described in the table below, there were a total of seven estimated sprints to complete the project. The following table describes the tasks scheduled to complete the project and their estimated completion time. Tasks are divided in groups and subgroups (yellow) and labeled in white. All the milestones are marked in green.

TABLE 1: PROJECT TASK LIST

	Name / Title	Goal	Type	Start Date	End Date	Days	Hrs
1	Back Office API		group	20/2	13/4	54	216
1.1	Software Design		subgroup	20/2	28/2	9	36
1.1.1	Development environment and CD/CI setup		task	20/2	22/2	3	12
1.1.2	UML design process		task	23/2	28/2	6	24
	Software development setup completed	1.3	milestone	28/2	28/2	1	4
1.2	Authentication and Security	1.3	subgroup	6/3	21/3	16	64
1.2.1	JWT auth module	1.3	task	6/3	14/3	9	36
1.2.2	Roles module	1.3	task	15/3	17/3	3	12
1.2.3	Team module	1.3	task	18/3	21/3	4	16
	System security completed	1.3	milestone	21/3	21/3	1	4
1.3	Model Process Workflow	1.3	subgroup	22/3	13/4	22	88
1.3.1	Project module	1.3	task	22/3	24/3	3	12
1.3.2	Machine learning model module	1.3	task	25/3	26/3	2	8
1.3.3	Machine learning state module	1.3	task	27/3	10/4	14	56
1.3.4	Machine learning prediction module	1.3	task	11/4	12/4	2	8
	Model workflow completed	1,3	milestone	12/4	12/4	1	4
1.3.5	R task scheduler	1.3	task	13/4	13/4	1	4
	BO admin system completed	1.3	milestone	13/4	13/4	1	4
2	Machine Learning R Scripts	1.1	group	14/4	5/5	22	88
2.1	Input R-scripts	1.1	task	14/4	22/4	9	36
2.2	kNN simple scripts	1.1	task	23/4	25/4	3	12
2.3	Output R-Scripts	1.1	task	26/4	28/4	3	12
2.4	R-Script process template completed	1.1	milestone	28/4	28/4	1	4
2.5	Naive Bayes single scripts	1.1	task	29/4	30/4	2	8
2.6	ANN single scripts	1.1	task	1/5	1/5	1	4
2.7	SVM single scripts	1.1	task	2/5	2/5	1	4
2.8	Decision tree single scripts	1.1	task	3/5	3/5	1	4

2.9	Random forest single scripts	1.1	task	4/5	4/5	1	4
2.10	R-Scripts completed	1.1	milestone	5/5	5/5	1	4
3	Blade Front-End	1.2	group	5/5	28/5	24	96
3.1	Blade Framework Setup	1.2	subgroup	5/5	28/5	24	96
3.1.1	Choose Bootstrap template	1.2	task	5/5	5/5	1	4
3.1.2	Base framework setup completed	1.2	task	6/5	6/5	1	4
	Integrate design to views	1.2	milestone	23/5	28/5	6	24
3.2	User Flow and Security	1.2	subgroup	7/5	11/5	5	20
3.2.1	Auth process	1.2	task	7/5	7/5	1	4
3.2.2	User profile UI	1.2	task	8/5	8/5	1	4
3.2.3	User management CRUD	1.2	task	9/5	9/5	1	4
3.2.4	Team CRUD	1.2	task	10/5	10/5	1	4
	User and security completed	1.2	milestone	11/5	11/5	1	4
3.3	Model Module	1.2	subgroup	12/5	19/5	8	32
3.3.1	Project CRUD	1.2	task	12/5	12/5	1	4
3.3.2	Model CRUD	1.2	task	13/5	13/5	1	4
3.3.3	Model State CRUD	1.2	task	14/5	15/5	2	8
3.3.4	Prediction CRUD	1.2	task	16/5	19/5	4	16
	Model module completed		milestone	19/5	19/5	1	4
	TOTAL					100	400

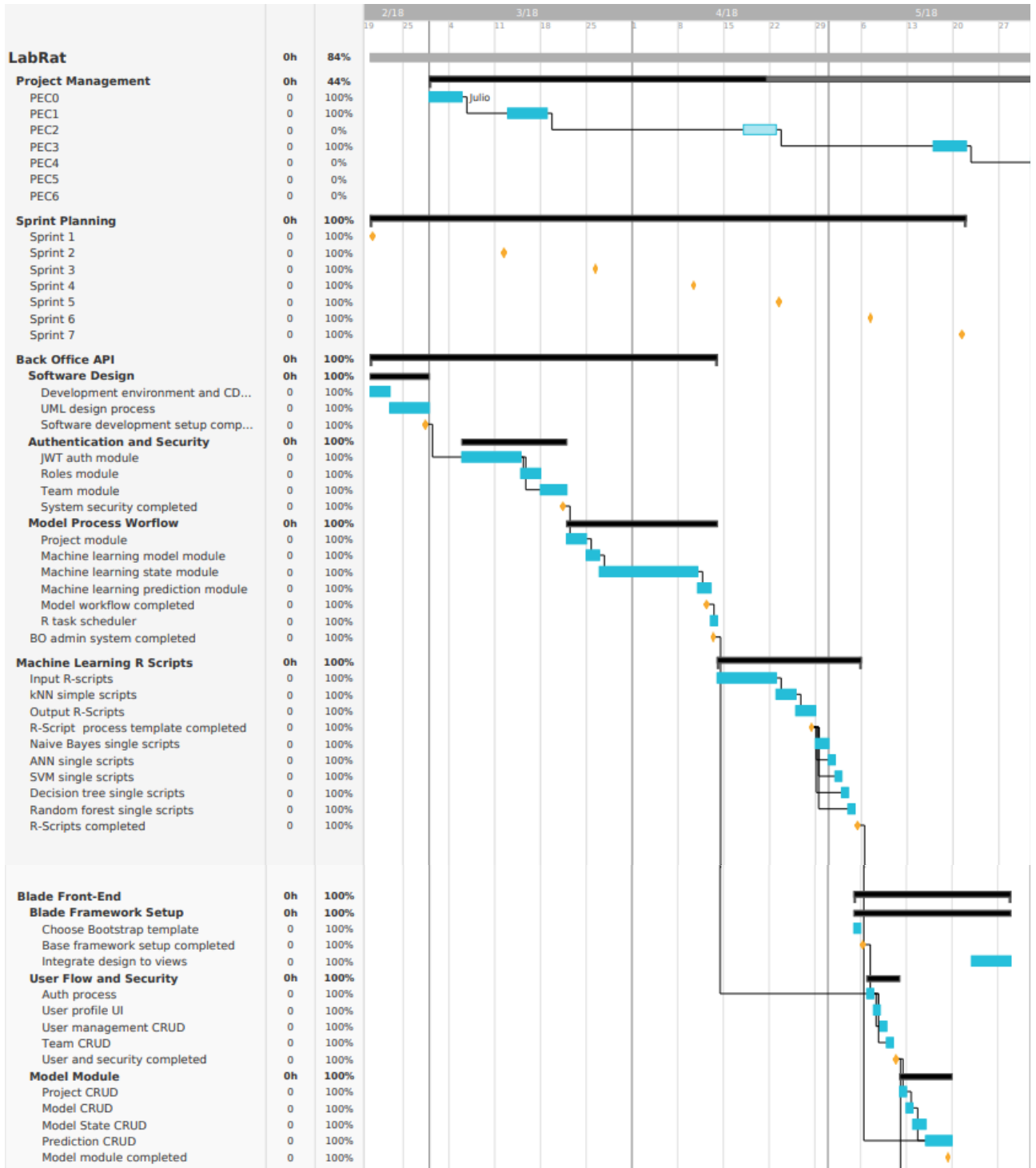


IMAGE 1: PROJECT GANTT DIAGRAM.

1.5 Deliverable brief description and summary

There are two main deliverable products as the result of the work done. These are the LabRat-Back-Office-API repository and the LabRat-R-Server:

- ❖ **LabRat-Back-Office-API**: it contains the necessary code for the back-office, the front-end and the API Rest implementation.
- ❖ **LabRat-R-Server**: This is the R server that allows for the SFTP data file transfers and the remote R scripting manipulation.

Both repositories can be found at <https://github.com/LabRatGroup>.

1.6 Remaining chapters and content brief description

The following chapters will discuss some of the most relevant technical details of the software infrastructure presented here. This will include the project general structure, the development process conventions used, a brief description of functional and non-functional requirements, a series of design UML and micro-service interaction diagrams, software requirements and essential installation requirements.

Also, a short risk analysis of the development process as well as a discussion of some the main challenges encountered along the development process will be discussed as part of the project memory.

Finally, a brief user manual on how to use the application.

2. Technical Design

Project structure

The following items describe the application development main steps and their description according to the planning groups and subgroups:

- ❖ **Back Office API:** involves the development of the back-office and API Rest tool.
 - ❖ **Software design:** this is the initial software design process necessary for every software development process. This section involves the elaboration of UML diagrams and the setup of the CD/CI and deployment environments.
 - ❖ **Authentication and security:** global setup for application security and user control necessary for API and machine learning model access.
 - ❖ **Model process workflow:** probably the most important section of the back office. This is where all the related services regarding machine model generation, editing and history control are developed. It also involves the unclassified data predictions services and manipulation. Services related to R scripts scheduling and performance are also developed in this section.
- ❖ **Machine learning R Scripts:** During this section all R scripts were developed for the selected machine learning algorithm. This is also where the data I/O mechanism from the back office into the R module, and vice-versa, are defined.
- ❖ **Blade/Vue Framework Setup:** This is where a design for the UI admin tool is defined and adapted to the Blade environment.
 - ❖ **Design integration:** The UI graphic design is integrated into the HTML code of the Blade front-end.
 - ❖ **User flow and security:** The security and route services are integrated into the front-end layout.
 - ❖ **Model Modules:** Development of the UI templates, services and API calls related to the project, model and prediction managers. This section also involves standard CRUD sections for each of the modules.

Development process

The development process was executed following modern software development standards. An Agile/Scrum methodology was followed using Jira to keep track of user stories in a two weeks length sprint periods. All project planning tasks previous to the Agile sprint planning was done at TeamGantt.

For the development coding activity PHPStorm was used as primary IDE tool. All the code was kept at GitHub, which integrated with CircleCI for

an effective testing process and CD/CI cycle with Heroku. An effective TDD programming technique, with end-to-end and unitary tests, was implemented to effectively implement the CD/CI process guaranteeing and effective and save deployment into the master and develop branches.

The production phase was carried out manually in a virtual server hosted by Digital Ocean and only after all the CD/CI cycle was successfully completed.

Functional requirements

There are two major areas to consider when defining LabRat functionality. The first and main area to discuss is the machine learning modeling and prediction capabilities of the system. LabRat allows the creation of machine learning trained models from raw data. When creating a model, the user can specify the machine learning algorithm to use and the desired parameters or range of parameters.

Furthermore, LabRat can also decide for the user when choosing the best algorithm for the trained model. The system can estimate the best possible model for data prediction.

Generated models can be improved by adding new additional training data to it and/or by modifying the training algorithm or its parameters to create a better and more accurate prediction model.

In addition, LabRat has a model state history which allows the user to rollback to previous models in case something goes wrong.

All the models are stored under a project-like structure, which can hold together several models from different datasets and algorithms. These trained models can later be used to make real predictions from unclassified data. All predictions are also stored under each model for further review.

The second area of interest in LabRat is the collaboration capabilities of the system. A user can create team structures and designate permissions to the team members that allow them to work on a specific project and, therefore, the models it may contain. Each team member, or individual user, can have specific permission upon a specific project models. It is also possible to add stand-alone users to a project without any previous team association.

Non-functional requirements

The system needs to be easy and cheap to maintain and also well decoupled. For such a purpose it is necessary to choose a reliable technology in terms of programming language and data storage methods as well as deployment costs and required hardware infrastructure.

In order to fulfill the decoupling characteristics of the system, the back-end and front-end must be developed as separated components. Updating any of these parts should not affect the other.

For the system to be able to adapt to any hardware infrastructure, the R-Caret scripting functionality should not be part of the API Rest or back-end. The main goal of this requirement is to be able to provide to the most CPU intense processes of the application of a separate server environment to run in if necessary. This will allow for an extra boost in performance in organizations where providing an extra machine for computer power is not subject to budget constraints. The system will be able to scale according to the organization machine power.

All R scripts should be able to be updated and improved regardless of the system input and output requirements. A modular or OOP paradigm should be implemented in R in order to promote the scalability of the scripts. Also, the connection between the back-end and the machine model scripting module should be independent of the technology used for machine learning analysis, allowing for a comfortable and painless switch from R to Python scripting if necessary.

UML model design

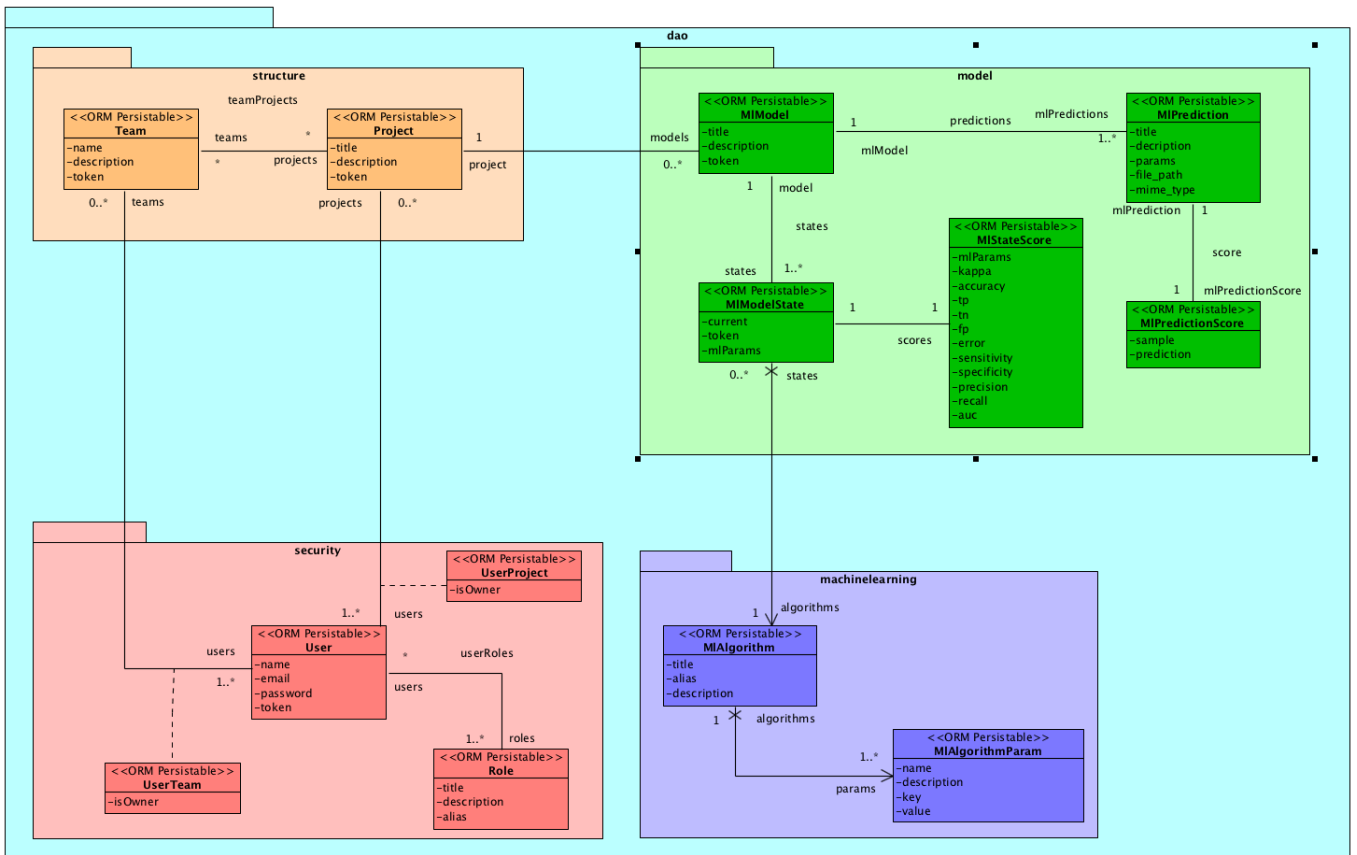


IMAGE 2: UML MODEL DESIGN

The system model map is divided in four different areas.

Structure

These models help contain the remaining models in the system. They are divided in two main models:

- ❖ **Team:** The *team* model groups users together to provide a security framework that defines who can access and manage the information stored in the system. When a team is created, it is assigned with a specific amount of users, this team is later relate to a project which can only be accessed by the users enclosed within the team. Every team has an owner user that can not be removed from the team.
- ❖ **Project** This *project* model contains the machine learning models and predictions. It is the main unit of work in the system.

Security

There are two major models within this category:

- ❖ **User:** The *user* defines a person identity and has restricted and monitored access to all the remaining areas within the system.

- ❖ **Role:** A *role* defines the function of a specific user and limit her/his actions within the application. The available roles are:
 - ❖ Staff member
 - ❖ Super administrator
 - ❖ Senior staff member
 - ❖ Junior staff member
 - ❖ Registered user
 - ❖ Regular registered user
 - ❖ Project Roles
 - ❖ Project administrator
 - ❖ Project manager
 - ❖ Project user

Users can be assigned to teams and/or projects. This means that a project access can be restricted to either users or team members. When a project is created by a user, she/he becomes the project owner and has full control over the project and its parts.

Machine Learning

The machine learning section contains two main models:

- ❖ **Algorithm:** The *mAlgorithm* model holds all the information related to the algorithms available for machine learning model training. These algorithms are given by default when the database is created and can only be modified via database query.
- ❖ **Algorithm parameters:** The *mAlgorithmParam* model holds information about all the different parameters used to tune up the specific algorithm used when training a model. Each algorithm can have one or more parameters. Each one of these *mAlgorithmParam* models hold information about the kind of values allowed (numeric, boolean or type) as well as the numeric range or possible values permitted. It also holds information regarding the default values used when automatically tuning up the algorithm for model training.

Machine Learning Model

These sets of models are used to hold information about the trained models and predictions. They are the core of the system when it comes to functionality. The main components are:

- ❖ **Model:** The model *mModel* defines a machine learning model to train. It encloses a specific study with predefined classes that will persist throughout the study.
- ❖ **Model state:** A model state (*mState*) is a trained model. It is called state because it holds information related to a specific set of parameters and data at a given time. There can be many states per model but only one is current or dominant. The dominant or current state can be switched as desired. Nevertheless, this is not recommended since the system always sets the current state to the one with the better performance score.

- ❖ **Model state score:** The *mIStateScore* model holds information about the performance of each of the states or trained model within a model. The *mIStateScore* model holds statistical information about the model performance such as kappa, accuracy, sensitivity and recall among others. This score defines which of states is current or dominant from the rest.
- ❖ **Prediction:** The prediction or *mIPrediction* model holds the necessary information to complete a prediction from a given dataset file. Each prediction uses the current or dominant state of the model that contains it to predict the class for each of the provided samples within. All the predictions within a model are recalculated whenever a new state proclaims itself current or dominant. This guarantees that the predicted data is always up to date with any trained model updates.
- ❖ **Prediction Score:** The *mIPredictionScore* model contains the results for a given prediction in terms of sample number or ID and resulting class.

Micro-service infrastructure

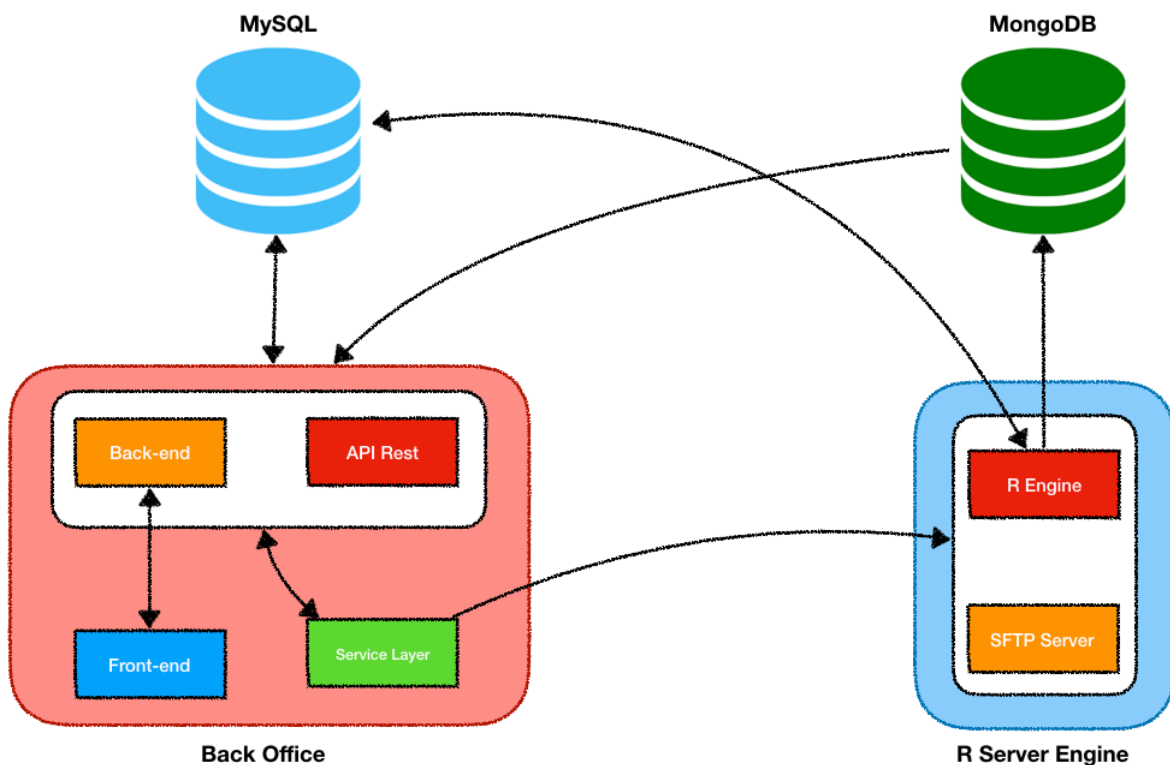


IMAGE 3: MICRO-SERVICE INFRASTRUCTURE

LabRat works as a micro-service infrastructure composed of two main elements.

Back-Office

This is the main application and consists of a admin or back-office tool manages the model training process and predictions. It is composed of:

- ❖ **Back-end:** This is the main set of controllers that interacts with the UI or front-end. It is the core of the MVC infrastructure.
- ❖ **Front-end:** It is the actual UI of the application built on top of the MVC layer.
- ❖ **Service layer:** This is where the business logic resides. It is compose by ORM models, repository and service classes. It is common to the back-end and API-Rest layers.
- ❖ **API-Rest:** This is the secure API-Rest layer that operates just as the back-end does. It is available through the port 80 via JWT. It shares the same service layer than the back-office MVC controllers.

R Server Engine

The R Server Engine is where all the statistical processes occur. It is built with the following components:

- ❖ **R Engine:** The R engine is built within a Node.js API server available at port 3030. It has two different entry points: one for model training and a second one for performing predictions.
- ❖ **SFTP Server:** The SFTP server is used to transfer training and prediction files from the back-office to the R serve engine. It seats over a Node.js server at port 3333. It stores the files within the same file system where the R engine is hosted, allowing direct access of these files by the R engine.

The back-end component uses a MySQL database to store states, predictions and scores among other relevant information related to teams, users and projects. It basically uses MySQL to store business logic related data that is structured and well defined.

Some of these models use JSON to store unstructured and robust data within MySQL. These involve mostly algorithm training parameters and default values as well as information regarding how these algorithm parameters are restraint.

All prediction score data is stored in MongoDB. Since a prediction can be compose by thousands of samples, a NoSQL database was chosen in order to improve performance when reading/writing large amount of data.

When a model is trained, the user uploads a CSV file composed of an usually large amount or columns, or predictors, and rows or samples. A training file can contain thousands of known samples used for model training. This files are very large in size and need to be easy to access. The files are uploaded (usually via intranet) to the SFTP server and stored within the same file system as the R engine. Having the SFTP server within the same code repository as the R engine ensures an easy access of those files by the R engine.

A similar process can be described for data prediction. The file with the data to be predicted is also uploaded via SFTP from the back-office to the the R server engine where it is stored at common grounds with the R engine.

It is important to mention that the R engine server can be hosted at a different server machine than the back-office server. This allows the system admin to use a high performance (and more expensive) machine to host the R server engine in order to provide more data process power to the machine learning processes. This why a micro-service infrastructure was chosen for this project.

Once the training or the prediction data has been uploaded, the back-office queue a training or a prediction job. This job executes an API call to the R engine server with a specific token that indicates R where in the MySQL database is the information needed to perform the requested task.

When a model training is requested, R queries information about what algorithms should use for building the models and where to find the training data file. Once the training process is completed, R writes the score information back to MySQL and informs the back-office queue job of the result. The process finalizes when the back-office service updates the job status.

When a prediction is requested, R queries MySQL for information about what model should be used to perform the prediction as well as the location of the prediction data file or samples. After the prediction is completed, the results are written into MongoDB. The back-office queue job is then notified and the job status updated.

If an error occurs during the training or the prediction process, R will not write back to the MySQL nor MongoDB database with score nor precision data; the back-office will then report the process as failed. It is therefore the back-office service who notifies the user about the job status according to the data found in both database servers.

Software installation requirements

For production purposes, the software is recommended to run under Linux OS systems (Ubuntu 18 recommended). The following list summarizes the minimal hardware and software requirements needed for production deployment.

- ❖ Ubuntu 18 Server under a 1 CPU/1 GB virtual machine with static IP v4 address and 25GB SSD.
- ❖ Apache 2
- ❖ PHP 7.1+.
- ❖ PHP mbstring extensión
- ❖ PHP mongoldb 1.4.2+ extensión
- ❖ Pear-PHP
- ❖ PHP Composer
- ❖ MySQL 5.6+
- ❖ MongoDB 3.5+
- ❖ Node.js 8.1+
- ❖ NPM 5.5+
- ❖ R-Base linux package
- ❖ Git linux package.
- ❖ Supervisor
- ❖ Forever

3. Risk Analysis and Development Strategies

Initial development risk analysis

There are several risk factors involved in the development of the project, most of them are time or technology related issues in some critical stages of the developing process. The following points were considered as risk factors at the beginning of the developing process:

- ❖ **Development delays due to R scripts I/O processes:** this item involves the development and planning of a specific strategy when it comes to preparing and reading the data from a CSV file into an R script and to a data base table. Extracting and storing model performance data and the model itself can be challenging if we wish to perfectly synchronize the back office tool with the R-Caret scripting service.
- ❖ **Complex ORM relations:** managing model states history and predictions can be difficult and could lead to developing issues and delays.
- ❖ **Learning curve:** some of the parts of the projects ought to be developed using new or uncommon technology in terms of my actual professional expertise. Therefore, it will be necessary to spend extra time learning and becoming familiar with those technologies. This may lead to delays in the actual project planning.
- ❖ **Front-end user experience:** when developing a front-end interface, there are many layers of “satisfaction” to be achieved in order to provide the user with a decent experience. Since the front-end development will be done at the end of the planning, chances are that some of the desired features, such as charts and configuration of model generation options; could be affected in terms of quality or removed in order to be able to close the project and provide the evaluation committee with a functional application.

Development strategies and troubleshooting

The following points described some of the major difficulties found along the development process and how they were resolved.

The file storage paradigm

The main challenge encountered in the development process was how to store the data files loaded into the back-office as well as the R model and prediction files resulting after each analysis. In order to resolve this challenge, the following requirements were considered:

- ❖ All stored files must be kept in a private server. Solutions such as an Amazon Bucket was discarded immediately.
- ❖ The storage ought to be persistent. This data is crucial for the system to be functional.

- ❖ File access must be fast. Since these files ought to be loaded for R analysis and back-office analysis result display, the file access latency should be kept low. It is important to keep in mind that these files could weight as much as 10Gb each.
- ❖ Files need to easily be transferred from the PHP back-office engine server to the R- Scripting server and vice-versa.

The first approach was to make use of MongoDB to store the files. MongoDB offers the possibility to store large amount of data per page and has available drivers for PHP and R.

Nonetheless, this approach led to a series of problems. The first challenge involved CircleCI not having native docker container support for Laravel+MongoDB. An average of four days were invested trying to figure out how to overcome this issues. The problem was resolved by forcing CircleCI to compile the Pecl driver for MongoDB on each test instance.

Compiling the Pecl extension in every test can be tedious, this problem led to the initiative of building a custom docker container for the project. Unless this initiative is not in the score of the project, it is indeed an improvement to be included in the future. At the current time, CircleCI is still compiling the driver every time the code is tested.

Although the MongoDB did solve the file exchange problem between the back-office and the R-Scripting engine, a second derived issue came along. The R driver for MongoDB did a lousy job when reading array data from MongoDB. While PHP was storing CSV data into an array of lines (a huge array as a matter of fact), R was reading those arrays as a single endless strings that needed to be parsed and cleaned. This was not acceptable at all.

The solution was, as usual, to keep it simple. By using the already embedded file storage feature in Laravel, files are now automatically stored via SFTP in an open-source Node.js based SFTP server included into the R-Scripting engine as a repo submodule. This solution allows Laravel to send the files to the R-Scripting engine server (regardless of its location) and for R to place the files in a common ground easily accessible for the back-office API tool. By hosting both platforms in the same server we can minimize the transfer lately mentioned before.

Laravel file storage method is code independent and ca be switched by simple modifying an environment variable.

Nevertheless, there is still a new challenge to be resolved and this is how the prediction results will be delivered back to the back-office API tool. This can be achieved in two ways:

Using SFTP transfer. This will be the natural solution to the problem. Laravel can download the files upon request with a low or no latency if operating under the same LAN.

Using MongoDB. If we store the prediction results in MongoDB, Laravel can make use of the noSQL engine advantages such as indexation and pagination. This option will also allow the R-Scripting engine to be hosted in a different LAN or remote server. Still, at this point of the development process I still do not know how will R store large amount of comma separated data into MongoDB. The final implementation procedure is still to be decided.

Nevertheless, I am confident that this decision will not impact the project planning since all the research needed to make a final decision has already be done.

This storage and CircleCI problems were mostly reflected in the model state and prediction related Jira tickets.

Front-end Alternative Plan

Although not mentioned in previous sections, originally, the front-end was meant to be developed in Angular.js. This became a risk factor after the file storage issue was resolved.

In order to prevent future inconvenience that could have placed the project completion deadline in danger, a quick switch to Blade/Vue was made. Since the API Rest was already developed, it was extremely easy to create new front-end controllers for Blade while keeping a fully functional API for third party applications to use.

One of the positive outcomes of the used of Blade as an integrated frontend solution is that the back-end server now comes in two flavors: one set of controllers for the Blade frontend and a mobile ready API set of controllers both working under the same service layer with almost no extra effort.

This API layer can be used via JWT for mobile integration or OAuth for third applications to make use of it.

DevOps deployment deviations

One of the major challenges faced during the development process was the deployment of the R server engine.

Originally, Heroku was chosen as the primary deployment server for development and production due to its simplicity and commodity when it comes to CD/CI processes.

Nevertheless, the R server engine required of two public ports in order for the back-office server to interacts with it. These are the port 3000 for the SFTP file server file upload and the port 3333 for the execution of specific model training and prediction tasks.

Heroku only allows for a single port to be exposed to the public (port 80) and it does not provide support for a multi-container intranet infrastructure. The PHP backend/frontend server could perfectly be stored in Heroku but not the R server and engine.

Several solutions were studied in order to overcome this limitation. The first approach was to use Node.js for subdomain/port redirection within the same container. This would allow the use of a single public port (port 80) and two different subdomains, one per each of the R server engine functionality (or internal port). This option was discarded since the SFTP server could not be easily manipulated as it is embedded within the application as an NPM dependency.

Another approach was to detach the R engine from the SFTP server. This solution implied the incorporation of a new TCP communication channel between both services in order to share data files. This solution would have had a huge performance impact in the data analysis and was therefore discarded.

The final, and definite solution, was to migrate the production code to the Digital Ocean hosting services. This solution has many attractive options to explore, these are:

- ❖ The use of a docker oriented predefined virtual server. This feature will indeed boost the idea of using a docker container for software deployment.
- ❖ The opportunity to scale the hardware in order to obtain a better performance server for R analysis.
- ❖ Money saving. Digital Ocean offers more memory and disk space for less compared with Heroku.
- ❖ Digital Ocean allows for a better server setup and management as they are non-volatile virtual machines.
- ❖ Although Digital Ocean does not offer an out-of-the-box CD/CI process, it can eventually be implemented using Kubernetes (not in the scope of this project at the current moment though.)

4. User Manual

Demo datasets

There are two test or demo datasets included in the project and accessible through the back-office UI. These datasets are available from the “Download Demo Data” link in the main menu. The data is normalized and prepared for machine learning monitored classification processing.

The two available datasets are:

- ❖ State of the art prediction of HIV-1 protease cleavage sites. Rognvaldsson et al. *Bioinformatics*, 2015, 1-7. doi: 10.1093/bioinformatics/btu810
- ❖ D Ayres de Campos et al. (2000) SisPorto 2.0 A Program for Automated Analysis of Cardiotocograms. *J Matern Fetal Med* 5:311-318

The first dataset comes with DNA orthogonal encoding and works really well with the kNN classification algorithm, although it can be used with any other available machine learning algorithm but at a higher performance cost.

The second dataset has two different data modalities. The first set contains raw numerical data while the second factorized data, which makes it more suitable for classification purposes.

Both datasets come with a research summary, a link to the complete research paper and a downloadable document with a manual machine learning analysis performed by the author of this research, that being Julio M. Fernández Jiménez.

A training and a prediction file can be found with each dataset group. They can all be used to test and compare the application results with the manual analysis provided with each study.

These dataset files can and should be used as template data files for other studies. Common column names such as “sample” and “result” should not be renamed in order for the R scripts to recognize them properly.

Project security

Despite the obvious user level security of the system, where a username and password is necessary to access the application, LabRart makes use of a permission system that restricts who can access a project and all the content stored under it.

By default, all created projects are private, and can only be accessed by the creator or project owner. The project owner can edit, delete, create and train models as well as to generate predictions within the project.

In order for the project owner to allow someone else to collaborate with a specific project, it is necessary to either directly add the user as a collaborator or as a member of a collaborator team.

To add a new user as a direct collaborator, it is necessary to access the project form (either when it is created or by editing the project). Once inside the project form, a new collaborator can be added through the collaborators section located at the right hand side of the form.

Add collaborators by email

ratke.cassandra@example.org Add new collaborator

Julio Fernandez jfernandez74@gmail.com	
Dimitri Beatty windler.jarod@example.org	Remove
Dr. Ila D'Amore II sarai.auer@example.com	Remove

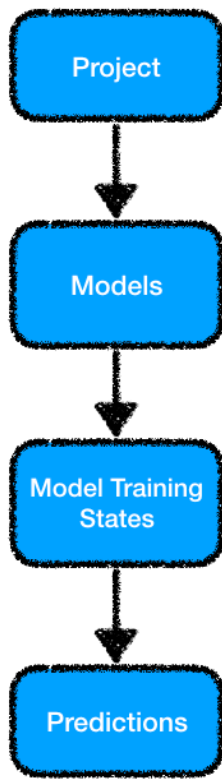
IMAGEN 4: COLLABORATOR FORM

To add a new collaborator, it is only necessary to provide the user email address and press the “Add new collaborator” button. If the a user with the provided email address exists, a new user block will appear in the collaborators section; a warning message will be display otherwise. To remove an individual collaborator from the project, just click the “Remove” button that corresponds the user you wish to remove.

Teams allow a product owner to assign access to projects to groups of users with just a single click. The project form has a multi-select HTML controller that allows the owner to select which teams will have full access to the project content.

The team creation process is similar to the individual collaborator form discussed above. Users are added by email to the team through the team form. Users can be added at anytime inheriting full access to all the related projects upon saving the team element.

System Workflow



Working with LabRat is all about workflow and hierarchy. It is important to understand how the data is structured before working with the system.

The main unit of work is the project. This is where we apply the access permissions and what sustains all the remaining and non security related data structures. A project can be compared to a study or research project.

A project is composed by models. A model is comparable to a dataset with a determinate number of predictors, classes and positive class.

A model can be trained as many times as we wish. Each model training intent is called a *state*. All states generated under a specific model are always stored and available within the system, but only the one with the highest score is used for the data prediction process. When a new state with a highest score is generated, a prediction event is triggered and all the predictions previously performed are recalculated again under the new and improved model.

IMAGE 5: APPLICATION DATA WORKFLOW

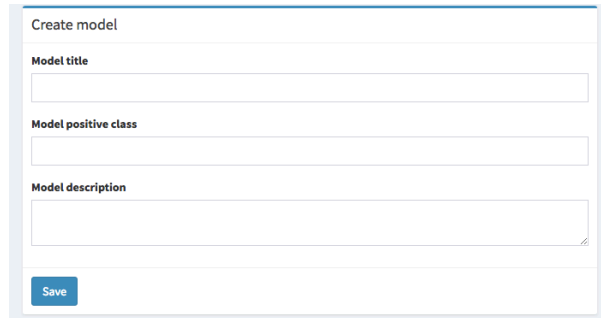
When a model is retrained, it is always done by adding extra training data rather than providing a similar dataset with a different (although better) structure. If we train a dataset with a file containing an extra predictor, classification parameter or data processed in a different, the predictions already available could fail on re-run if this model turned out to have a better performance than the others.

If you, as a researcher, has a more suitable dataset for a specific model in your research, it is highly recommended that you create and train a completely new model with this special characteristics rather than trying to improve or overwrite an existing one with non compatible prediction data.

Each model can have as many predictions as you wish, this predictions are always executed with the higher scored state found under its model. It is necessary to have at least one trained model or state to run a prediction.

Model training

When training a model, it is first necessary to create a model object. The model creation process is simple. Only the title and the positive class is needed.



The image shows a web form titled "Create model". It contains three text input fields: "Model title", "Model positive class", and "Model description". A blue "Save" button is positioned at the bottom left of the form area.

IMAGE 6: MODEL FORM

Once a model has been created, it has to be trained in order to run prediction jobs against it. The training process consist in completing a form that defines which algorithm will be used and what parameters to use with it.

The algorithm can be chosen by selecting from the select HTML component. When an algorithm is selected, a new set of parameters are described underneath. These parameters are autofilled with standard predefined values that can be modified as needed.

Besides the algorithm natural parameters, the system allows for the selection of a data preprocessing option, a resampling method and its iterations and what performance option to use for scoring the algorithm.

If you have enough CPU power in your R engine server, you can opt for the system to "Use the best cited method for you". This will iterate through all the available algorithms and options and select the one with the best score.

After all the analysis options have been defined, a CSV file with the training data must be attached for model training. If a model is run for a second time with the same data but different options, the data file can be omitted since LabRat will always use the last supplied data file for model training.

All model states are always available for the user to review. The can be accessed under the model detail section.

ID	Created	Algorithm	Status	Performance
12	2018-06-03 20:11:55	k-Nearest Neighbor	Active State	Accuracy: 90% / Kappa: 0.4
11	2018-06-03 20:07:22	k-Nearest Neighbor	Failed	Failed
10	2018-06-03 20:05:20	k-Nearest Neighbor	Failed	Failed
9	2018-06-03 20:00:24	k-Nearest Neighbor	Success	Accuracy: 90% / Kappa: 0.4
8	2018-06-03 19:57:12	k-Nearest Neighbor	Failed	Failed

IMAGE 7: MODEL STATE LIST

Train model

Training algorithm selection

Please select the desired training algorithm

k-Nearest Neighbor

Number of Neighbors (*k*)

From: 2 To: 5 Steps: 1

min: 1, max: 10, steps: 1

Data preprocessing options

Please select the desired preprocessing method

center

Resampling optimization method

Please select the desired re-sampling method

k-fold cross-validation

Number (10)

min: 1, max: 20, steps: 1

Data performance options

Please select the desired metric to calculate performance from.

Accuracy

Training data upload

Please select file containing the data you wish to use to train the model.

Seleccionar archivo Ningún archi...seleccionado

Train model

IMAGE 8: MODEL TRAINING FORM

Data prediction

A data prediction can only be requested under an already trained model. To predict data, it is only necessary to provide the job title and the data file with the sample names or description as well as the predictor fields.

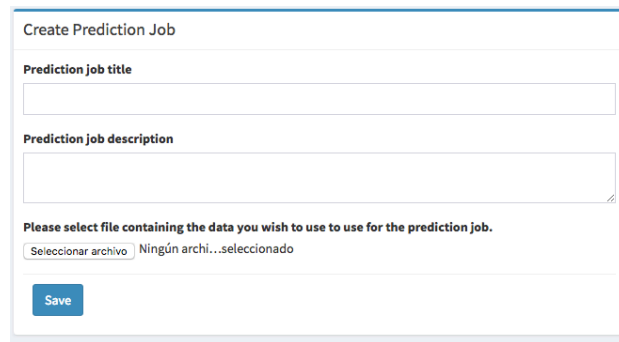


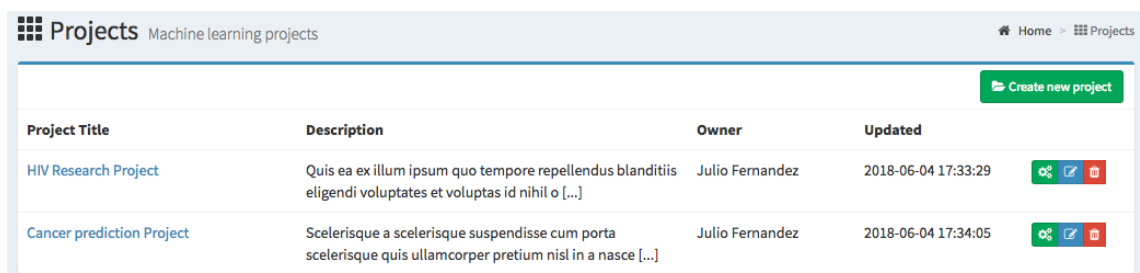
IMAGE 9: DATA PREDICTION FORM

User graphical interface

The user graphical interface can be navigated using the application workflow defined above. This means that the models can be found under the project details sheet and the predictions under each model detail section. Also, and in order to create or modified any component (such as a model or a prediction) we must go first to the detail sheet of the element that contains it. That is, if we wish to create a new prediction, we must first find the model used for such a prediction that can be found under that project that contains it.

In order to ease the navigation process, it is also possible to access all available models and prediction outside of the scope of their corresponding projects. Still, every action performed under any of this elements (such as editing or training) will lead us back to the hierarchical user interface model oriented structure.

The user interface follows a CRUD like convention. The following image illustrate the basic project list.



Project Title	Description	Owner	Updated
HIV Research Project	Quis ea ex illum ipsum quo tempore repellendus blanditiis eligendi voluptates et voluptas id nihil o [...]	Julio Fernandez	2018-06-04 17:33:29
Cancer prediction Project	Scelerisque a scelerisque suspendisse cum porta scelerisque quis ullamcorper pretium nisl in a nasce [...]	Julio Fernandez	2018-06-04 17:34:05

IMAGE 10: PROJECT LIST

On the top of the image we have the section title and breadcrumbs that can be used to navigate backwards and to have a point of reference that indicates us where we are now.

Each list element has a title, that leads to the detail view (where we can usually find other child elements such as models and predictions), and a toolbar with actions that can be performed over the listed element.

Economical Evaluation

Although this an open source project, it can be valued in terms of cost s involved in the production process and annual maintenance. The following tables summarized such expenses. Each of the cost/hr specified is related to the task category described within the table. All the cost/hr values used below are an approximation to the current European professional market according to my own personal experience as a project manager for the professional freelance market.

Production costs

Description		Hours	Cost/hr	Total
Back-office API	Back-end Engineer	216	40,00 €	8640,00 €
Machine Learning R Scripts	Biotechnician	88	75,00 €	6600,00 €
Blade Front-end	Front-end Engineer	96	60,00 €	5760,00 €
Project management	Project manager	35	30,00 €	1050,00 €
				22050,00 €

TABLE 2: PRODUCTION COSTS

Annual fixed costs

Description	Cost/month	Total
Back-end hosting	20,00 €	240,00 €
R engine server costing	80,00 €	960,00 €
Domain name registration	2,00 €	24,00 €
		1224,00 €

TABLE 3: ANNUAL FIXED COSTS

Product Viability

LabRat is totally viable for commercialization or the open market since it covers a necessity already demanded by the scientific community. Nevertheless, it is realistic, as well as necessary, to mention that the application needs some minor improvements in the following areas before it can be released as a fully functional and commercial project:

- ❖ **Usability:** The usability of the system has not be tested yet. It is important to test and contrast the product usability with at least two different sources before it can be released as a function solution.
- ❖ **Deployment:** As it will be mentioned in the conclusion's section, the production deployment process must to be automatized using Docker containers.
- ❖ **Maintenance:** As explained in the conclusions, a code maintenance and distribution plan needs to be implemented to guarantee its quality.

5. Conclusions

There are many practical and academic conclusions that can be extracted from the development and project managing experience processes involved during the making of this application.

From a project management point of view, the use of the Agile/Scrum methodology for the development of a project with a single and unique workforce unit was proven to be effective.

It is a fact that I had my doubts about being the project manager and the only developer involved in this project. Even though some licenses were taking at the end of the project when it comes to project management (the use of shorter user stories, longer sprints or the disposal of several non-crucial Jira tickets) a strong and solid Scrum methodology was followed without hesitation.

The Agile discipline imposed from the very beginning, was indeed a pillar of confidence when time and due work seemed not to be getting along. The fact that there was a complete project planning in Jira, and as a Gantt diagram, became a powerful planning prediction tool.

As a developer, one of the main achievements has been learning how to implement a micro-service system design pattern with asynchronic calls using a synchronic programming language.

In order to get PHP and R to work together without loosing the scaling capabilities of the system statistical analysis component, a micro-services system had to be implemented. This was accomplished thanks to the effectiveness of the Laravel framework, which allows the implementation of asynchronic queued jobs, Node.js and the Express module for HTTP server creation. These three technical elements, along with some of the Laravel embedded libraries for HTTP calls, helped in the construction of the micro-services environment.

On the downside, Laravel turned out to be inefficient in the elaboration of the basic CRUD calls for the API Rest and service layers. This could have been easily resolved with Loopback, which was not involved in the project as explained in previous sections. Nevertheless, Loopback will be seriously taking into consideration in future projects.

Every development tool was designed for a specific use and need. An efficient piece of software should implement what best suits its needs without any technological limitations.

Goals Analysis

All proposed goals were effectively accomplished at project completion. The following reflexion summarizes the facts that lead us to confirm the completion of the proposed sub-goals.

(1.1) To minimized custom made scripting tasks for machine learning analysis.

This goal was accomplished by the completion of the R scripts that processes the submitted data in order to provide a trained model, which is later on use to make predictions over unknown or unclassified data.

The scripts were coded considering the generic nature of the data and the different, but predictable, deviation of its nature. Facts such as the number of classes, predictors, or even the possibility of predictions errors due to corrupted algorithm parameters were considered in order to bullet prove the training and prediction phases.

In order to accomplish this goal, several datasets from different sources and nature were fed into the system, which was programmatically adapted in terms of the different patterns and behavioral characteristics from each dataset.

(1.2) To provide a machine learning trained model environment where the scientific community can share and use multidisciplinary trained models on demand.

This goal was accomplished by the implementation of the back-end server and all of its components which involved the front-end design and implementation, the API Rest interface and the service layer.

This back-office tool provides several environments (MVC and API Rest) where the scientific community can create and share trained models through the internet or private intranet.

(1.3) To provide a private and encapsulated environment for machine learning based data analysis and storage.

Since LabRat was develop to be used as a private environment, the nature of the application infrastructure itself complies with the requirement specified in this goal.

The open-source nature of the project and the capability to be installed as a private application within a controlled environment accomplishes the private and encapsulated working environment required by this sub-goal.

The effective accomplishment of all of the propose goals lead to effectively state that the delivered project has indeed **provided the scientific community with a collaborative open source application**

to generate machine learning trained models and predictions from unclassified data.

Methodology and Planning Challenges

The planning method adopted for this project was indeed the right one to complete the proposed goals. The Agile process methodology has fulfilled the development method expectations as it allowed to modify the proposed planning as needed.

Although the planning was very well defined from the very beginning, changes had to be made to adapt to the new challenges that the project requirements imposed along the development process.

Some of these challenges, such as the implementation of the dual port for the R engine server and data storage problem, added extra time to the development process. On the other hand, the R scripts regarding the implementation of each individual algorithm, were completed before schedule, which helped keeping a balance between the extra time added to the project and planning and the time saved as the result of over-scoped tasks.

One of the major planning decisions that we were taking, in spite of the correct adjustment of the project management task, was to switch front-end technologies. When the initial planning was presented, Angular.js was chosen as primary and unique front-end technology along with the back-end API Rest presented in the discussion of this project. Nevertheless, my professional experience as a developer made me decide to switch technologies in order to prevent unnecessary delays as a consequence of facing a steep and unnecessary learning curve that this technology could have brought along during the last stages of the development process.

This decision had two negative side effects. The first one was the implementation of a new design pattern, the MVC pattern, which was not initially scoped and could have brought delays in the project planning; the second was a lesser usability of the user interface due to the constraints the new implemented technology (Blade/Vue) had.

Nevertheless, none of these risks turned out to have a negative impact in the project planning as it was completed a week ahead of schedule.

Improvements and future software development guidelines

Software development is an ongoing process of perfection always oriented to provide the client with the best experience. The following items describe some possible, and sometimes necessary, features to be implemented for the next released version:

- ❖ **Dockerized deployment:** The production deployment process can be problematic and complex, specially when deploying at a fresh environment. This is why it is imperative to include a Docker box within each of the application repositories. This will simplify the deployment process. The goal is to reduce the deployment process to the execution of a simple bash command that will create, run and expose the all the needed docker images as it synchronizes the code from the repository master branch.
- ❖ **CD/CI:** Since a dockerized deployment is scheduled for the next software release, a continues development and integration strategy must be implemented. This will improve the application in the following areas:
 - ❖ **Client satisfaction:** The CD/CI process will be implemented into the repository Docker container, allowing the software to update when a new merge is detected in the master branch. This can be extremely convenient for clients who decides to clone and install the software into their private machines for local use. Every client hosting an instance of the application will automatically have the system updated with no extra work. Nevertheless, this will be an optional feature for the end user and only available whenever a manual update is not required.
 - ❖ **Smooth development:** The use of Docker as a way to implementing a CD/CI process will lead to a better development process by creating a new virtual machine for software testing of individual branches before merging into develop or master. This will replace the Heroku hosting completely as the development process migrates into a more enterprise-like environment. So far, Kubernetes seems like the obvious choice for in-the-house CD/CI development.
- ❖ **Data preprocessing:** At the current moment, all the data used for model training and prediction must be somehow pre-processed and modified before it can be used. This requires for the user to have a notion of R or still require of a bioinformatician to complete this process. In order to better automatized the process, a new section will be implemented into the model training process and prediction where the user can decide how to treat the data columns before proceeding.
- ❖ **Charting:** Since the system allows for the use of different parameters when training a model, it is possible to display performance charts of each of the models handled by R before choosing the one with the best performance. The model detail sheet will therefore include a chart section where the user can compare and observe the evolution of each model for each of the provided parameters.
- ❖ **Notification processes:** This new feature includes improvements in the UI and data flow. The user will be notified via email and web the moment a model has been trained or a prediction is completed without refreshing the browser. This will improve the user experience and the usability of the system.
- ❖ **Automatic prediction tuneup:** When choosing the right algorithm for the training data, the system uses a set of predefined parameters per

algorithm tested by the system. These parameters can not be modified by the end user through the web interface. A new feature will allow the user to manipulate these parameters according to her/his needs and hardware power available.

- ❖ **internationalization**: Although the project text has been optimized for internationalization purposes, it has not been yet translated into other languages besides English. This is indeed a improvement to be completed in future updates.

6. Glossary

Angular.js	A JavaScript framework for front-end programming.
API	Defined as an application programming interface, it is a set of commands that allows the interaction with a piece of software in order to execute code.
AWS	Amazon Web Service. It is the Amazon commercial server infrastructure.
CD/CI	Continuous development and integration. It can be defined as a set of steps or software development procedure that allows the code to be developed, tested and integrated into the main production code automatically.
Component diagram	A software engineer diagram that shows the relations and interactions between the different parts of a piece of software and how they communicate with each other.
Digital Ocean	A virtual machine service for software deployment
Framework	A software development tool that allows for the development of code within a set of predefined tools and conventions while simplifying the development process and allowing the developer to focus on the actual code. It takes care of the common code that interacts with database transactions, string manipulation, model creation and other general development tasks.
Git	A software version control service for keeping track of software development and integration.
Git-Hub	A comercial Git service for code hosting.
Heroku	A hosting and CD/CI platform for software development.
Laravel	Framework for software development.

Micro-service	A structural design pattern for software development consisting in the development of small and independent pieces of software that work together as a single unit.
MongoDB	NoSQL database engine for storing unstructured data.
MVC	A structural design pattern that separates the code into a three later structure: the database model, the controller logic and the user interface view.
MySQL	Relational database engine for storing structural related data.
Node.js	Server side programming engine based on JavaScript and usually used for server communication programming purposes.
PHP	A multipurpose programing language for the web based on C and C++.
PhpMyAdmin	A web based MySQL admin tools.
R	Statistical and mathematical programming language.
Repository	A section in Git where the code is stored.
REST	A software communication design paradigm that allows a piece of software to communicate to another through regular web-like URL addresses in order to execute a specific software functionality.
SaaS	Software as a service. It is a software distribution model that is remotely hosted at a server or cloud service. The end-user can access it via internet. It does not require any maintenance nor installation process for the user to make use of it. It can be described as a service specific web application with licensing and specific use restrictions. ^[25]
Spring Boot	A java based development framework for enterprise software development.

Symfony	A PHP based framework for enterprise software development.
UML	Unified model language, used for software model and database design.
Vue	A JavaScript framework for complex UI design and user interaction.
Web application	It is a more generic and general concept that englobe any service hosted remotely and used by a client via browser. A SaaS can be considered to be a web application but with licensing and other legal restriction that may involve maintenance and use restrictions.

7. Bibliography

- [1] CircleCI: <https://circleci.com/gh/LabRatGroup>
- [2] Sentry: <https://sentry.io/>
- [3] Bugsnag: <https://bugsnag.com>
- [4] Laravel Mongo Driver: <https://github.com/jenssegers/laravel-mongodb>
- [5] How to install PHP (7 or 7.2) on Ubuntu: <https://thishosting.rocks/install-php-on-ubuntu/>
- [6] Uninstall mongodb php driver and install a different version: <https://stackoverflow.com/questions/24116235/uninstall-mongodb-php-driver-and-install-a-different-version>
- [8] Run rJava with RStudio under OSX 10.10, 10.11 (El Capitan) or 10.12 (Sierra): [https://github.com/MTFA/CohortEx/wiki/Run-rJava-with-RStudio-under-OSX-10.10,-10.11-\(El-Capitan\)-or-10.12-\(Sierra\)](https://github.com/MTFA/CohortEx/wiki/Run-rJava-with-RStudio-under-OSX-10.10,-10.11-(El-Capitan)-or-10.12-(Sierra))
- [8] Install MongoDB Community Edition on macOS: <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/>
- [9] Using MongoDB with R | DataScience: <https://datascienceplus.com/using-mongodb-with-r/>
- [10] Shell.js cheatsheet: <https://devhints.io/shelljs>
- [11] shelljs/shelljs - Portable Unix shell commands for Node.js: <https://github.com/shelljs/shelljs>
- [12] tkambler/sftp-server: A Node.js-based SFTP server with an integrated REST API for querying users / files.: <https://github.com/tkambler/sftp-server>
- [13] Tuning Machine Learning Models Using the Caret R Package: <https://machinelearningmastery.com/tuning-machine-learning-models-using-the-caret-r-package/>
- [14] How To Estimate Model Accuracy in R Using The Caret Package: <https://machinelearningmastery.com/how-to-estimate-model-accuracy-in-r-using-the-caret-package/>
- [15] mongodb - PHP, mongo.so failed to write: Stack Overflow - <https://stackoverflow.com/questions/33640553/php-mongo-so-failed-to-write>

- [16] **preProcess function | R Documentation:** <https://www.rdocumentation.org/packages/caret/versions/6.0-79/topics/preProcess>
- [17] **Laravel-AdminLTE:** <https://github.com/jeroennoten/Laravel-AdminLTE>
- [18] **Sisporto 2.0: A program for automated analysis of cardiocograms:** [https://onlinelibrary.wiley.com/doi/abs/10.1002/1520-6661%28200009/10%299%3A5%3C311%3A%3AAID-MFM12%3E3.0.CO%3B2-9](https://onlinelibrary.wiley.com/doi/abs/10.1002/1520-6661%28200009%10%299%3A5%3C311%3A%3AAID-MFM12%3E3.0.CO%3B2-9)
- [19] **State of the art prediction of HIV-1 protease cleavage sites:** <https://academic.oup.com/bioinformatics/article/31/8/1204/212810>
- [20] **¿Cómo instalar MongoDB en Ubuntu 16.04?:** <https://www.digitalocean.com/community/tutorials/como-instalar-mongodb-en-ubuntu-16-04-es>
- [21] **How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 18.04:** <https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-ubuntu-18-04>
- [22] **How To Set Up a Remote Database to Optimize Site Performance with MySQL on Ubuntu 16.04:** <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-remote-database-to-optimize-site-performance-with-mysql-on-ubuntu-16-04>
- [23] **Laravel 5.1 404 not found on apache server:** <https://stackoverflow.com/questions/39178272/laravel-5-1-404-not-found-on-apache-server>
- [24] **¿Cómo Instalar y Proteger phpMyAdmin en Ubuntu 16.04?:** <https://www.digitalocean.com/community/tutorials/como-instalar-y-proteger-phpmyadmin-en-ubuntu-16-04-es>
- [25] **Software como servicio:** https://es.wikipedia.org/wiki/Software_como_servicio
- [26] **How To Install and Manage Supervisor on Ubuntu and Debian VPS:** <https://www.digitalocean.com/community/tutorials/how-to-install-and-manage-supervisor-on-ubuntu-and-debian-vps>
- [27] **PHPStorm:** <https://www.jetbrains.com/phpstorm/>
- [28] **Lipsum Pro:** <http://lipsum.pro/>
- [29] **JSONLint - The JSON Validator:** <https://jsonlint.com/>

[30] foreverjs/forever: <https://github.com/foreverjs/forever>

[31] How to Install and Secure MongoDB on Ubuntu 16.04: <https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-mongodb-on-ubuntu-16-04>

[32] How to Install R Packages using devtools on Ubuntu 16.04: <https://www.digitalocean.com/community/tutorials/how-to-install-r-packages-using-devtools-on-ubuntu-16-04>