Development, Validation and Risk-Prevention of an Application with ML features
Wine Quality Prediction web-service case study

Lucas Gisselaire

Université Grenoble Alpes

Abstract. In this report I detail the different steps I took to develop, validate and secure an application with artificial intelligence features. The objective is to create a software system with real-world applications and finding ways to validate so as to prevent the unpredictability induced by the machine-learning features to cause problems for the user.

Keywords: Software Development · Machine Learning · Software Verification · Trace Analysis · Risk Prevention

1 Introduction

Nowadays, more and more software systems and solutions are making use of machine-learning algorithms to make decisions with the end-goal of undertaking some actions. A wrong prediction made by the system can have dire consequences on the user, sometimes going as far as endangering human life.
In this context, it seems clear that being able to verify the correctness of such applications is of good importance. However, due to the unpredictable nature of machine-learning, there is no known way to certify whether a given prediction is correct or not with 100% accuracy. With that in mind, it is also important for software systems to be able to mitigate the risks induced by this seemingly unavoidable unpredictability.

This document is a report of my internship done as part of a collaboration between Ms. du Bousquet (Grenoble's LIG - Software Validation) and Mr. Nakamura's (Kobe's Institute of Engineering - Software Systems) teams trying to deal with this problematic. In section 2, I will go over the different methods I researched for software validation. In section 3, I'll explain the way to go about developing ML models and web services using Microsoft's Azure Machine Learning Studio solution. Then in section 4, I will go over how I made my case study and detail the methods I used to validate and prevent the risks I recognized.

2 Validation of machine-learning software and models

In this section, I will go over some methods that can be used to validate or facilitate the validation of software with learning features.

2.1 Improving testability of ML software

Context Artificial Intelligence (AI) is becoming more and more prevalent in software systems. That being said, software & human security problems are regularly being reported, which means that there is a real need to extensively verify and test these systems. It is however particularly difficult to validate software that uses AI techniques. The article *Improving Testability of Software Systems That Include a Learning Feature*[1] reports the way its authors have modified the design of a given system so as to facilitate its validation as well as what they've learned thanks to this. The provided case study was made on an intelligent AC system.

Validation of Machine Learning algorithms There are multiple criteria that can be used to validate an algorithm that uses *ML* (Machine Learning) techniques:

– *Performance*: Validation of predictions made after the initial learning phase
– *Stability*: Quantitative evaluation of how much the algorithm is disrupted by comparatively small changes in input's structure
– *Training Speed*: Evaluation of the initial training's speed

Other properties/metrics can also be used for different algorithms or to compare algorithms together, such as *efficacy* or *tightness*.

In the same fashion, there also are different methods to evaluate the quality of ML algorithms:

– *Holdout Method*: The data available for initial training is split in two: one part is used for training and the other for validation
– *Cross-validation*: The dataset is split in k groups of instances of same size. One of these groups is used for validation

These approaches aren't really fit for real-life applications where the initial learning is done ”on the fly” and where dataset size and time constraints make these methods intractable.

Validation of the final system Because of the generally evolving nature of systems that implement ML algorithms, it is often impossible to get precise environmental characteristics. A way to proceed is to make use of simulations and to compare their results. It is also possible to go forward with a model verification. Whatever the means, the goal is to be able to exhibit software properties (such as security of robustness) relevant for verification. It is however still difficult to properly test the final system on which we shouldn't be able to control inputs; this makes simulations even more appealing.

Testability and software engineering Testability defines the system's testing capabilities. It is a growing need in the industry. Increasing the testability often consists in increasing the *observability* (being able to observe the inner workings of the system) and/or increasing controllability (being able to control the system).

2.2 Trace analysis using ParTraP

Introduction

ParTraP is a lightweight formal verification language designed to be used by engineers with no background or knowledge in formal methods. Its use consists in writing special expressions called ”properties” that allow to verify that the state at a given point in the execution of a program is correct. The article *Environment for the ParTraP trace property language (tool demonstration)*[2] serves as a demonstration of the language and its capabilities. The ParTraP programming environment is an Eclipse plugin, and it uses JSON-formatted trace files.

Generating ParTraP traces in a software application

As stated in the previous section, ParTraP requires a trace file to test properties. In order to be compatible with ParTraP, the execution of a program should thus produce a trace compatible with ParTraP. For that purpose, I modified an IoT application I made in a pervasive computing course to to account for this. The trace is produced as the program executes, and is output when the user asks for it. With such a trace, I am able to verify different properties of my application. To generate the actual file, I used built-in Java JSON bindings.

ParTraP-compliant trace format

ParTraP uses JSON-formatted files as input traces. In order to be compliant with ParTraP, a JSON trace should repect multiple criteria:

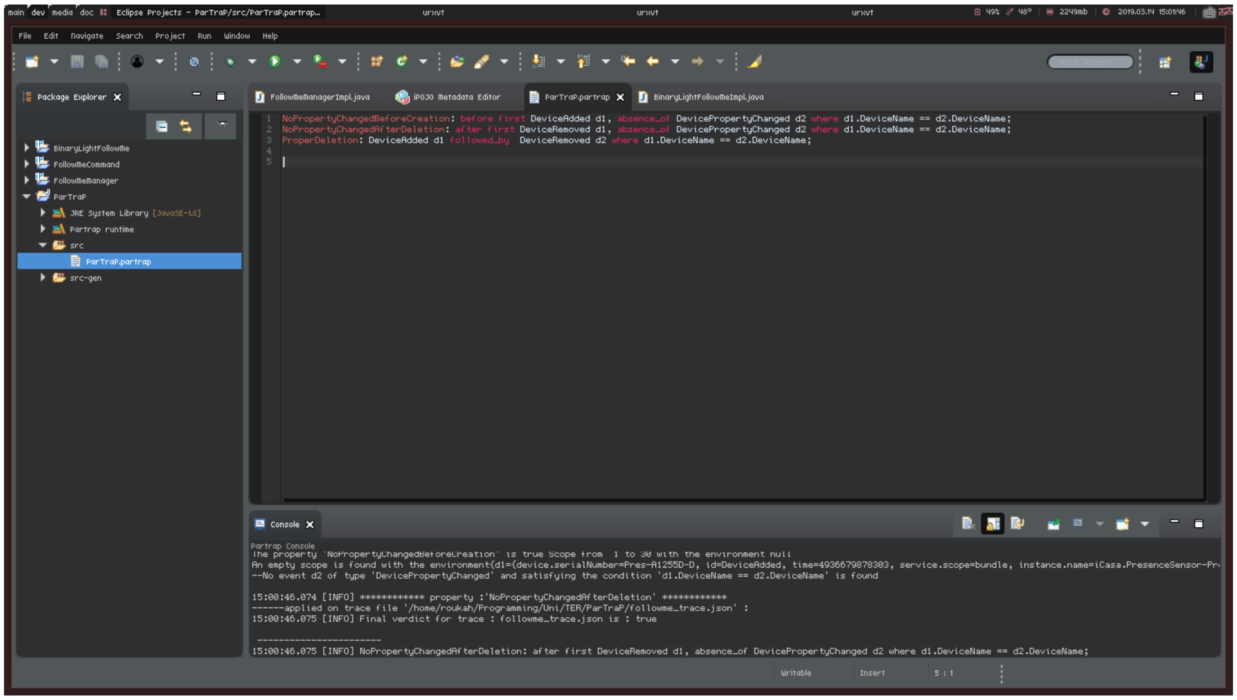
– A ParTraP trace is a single, ordered list of collections (array of objects).
– Each object of this collection is called an "Event". All properties will be tested on these events.
– All events have an identifier and a timestamp. They can be shared between multiple events.
– Events can have parameters, which are key/value couples. These parameters are records of the execution state when the event took place.

ParTraP property syntax

ParTraP uses its own syntax to test properties on traces. A basic property (that I will use as an illustration) is the following:

```
FirstProperty: abscence_of Error e where e.cause == "EoF"
```

– Properties are uniquely identified by a name: *FirstProperty.*
– Properties match a subset of the events of the trace using a "descriptor". Here, the descriptor matches all events of ID *Error* that have a condition on the *cause* parameter, expressed in the *where* clause.
– The above property can thus be interpreted as: "Verify that there is no Event of ID 'Error' with a parameter couple of key 'Cause' equal to the value 'EOF'"
– Python can be embedded in the ParTraP environment, making it possible to use Python expressions in properties.



```
NoPropertyChangedBeforeCreation: before first DeviceAdded d1, absence_of DevicePropertyChanged d2 where d1.DeviceName == d2.DeviceName;
NoPropertyChangedAfterDeletion: after first DeviceRemoved d1, absence_of DevicePropertyChanged d2 where d1.DeviceName == d2.DeviceName;
ProperDeletion: DeviceAdded d1 followed_by  DeviceRemoved d2 where d1.DeviceName == d2.DeviceName;
```

```
Partrap Console
The property 'NoPropertyChangedBeforeCreation' is true Scope from  1 to 30 with the environment null
An empty scope is found with the environment{d1={device.serialNumber=Pres-A1255D-D, id=DeviceAdded, time=4936679878303, service.scope=bundle, instance.name=iCasa.PresenceSensor-Pr
--No event d2 of type 'DevicePropertyChanged' and satisfying the condition 'd1.DeviceName == d2.DeviceName' is found

15:00:46.074 [INFO] ************ property :'NoPropertyChangedAfterDeletion' ************
------applied on trace file '/home/roukah/Programming/Uni/TER/ParTraP/followme_trace.json' :
15:00:46.075 [INFO] Final verdict for trace : followme_trace.json is : true

 ----------------------
15:00:46.075 [INFO] NoPropertyChangedAfterDeletion: after first DeviceRemoved d1, absence_of DevicePropertyChanged d2 where d1.DeviceName == d2.DeviceName;
```

Fig. 1. A basic example of ParTraP properties in the ParTraP environment

A more extensive explanation of ParTraP properties can be found in the ParTraP thesis paper[3].

Using traces to validate ML applications In the context of a system that uses a ML-based predictive model to make decisions, it is possible to imagine using trace properties as a tool to assist the validation of the "consistency" of the model. Some uses I can think of include for example checking that the input data of the model is the same as the data the user sent, or ensuring that if the same input is fed twice to a deterministic model then the same output is returned.
The point is, even if traces wouldn't allow to certify the quality of a prediction, they can still be used to validate the consistency of predictions.

2.3 Predictive models

A predictive model is a model that tries to predict an outcome using statistics. The value of the outcome can either be a number (regression) or a label (classification). We are interested in applications which make use of predictive models, that is applications which actions depend on the outcome of a predictive model for a given input.

Choosing the right model There is no systematic way to choose the algorithm to use for predictive models, as both the training data (data used to create the model) and input data (data fed to the model to guess an outcome) are not known in advance and they dictate the quality of predictions.
Some criteria to base the algorithm choice off of depends on the size of the data, its quality (coherence, "cleanliness"), the number of features, the desired accuracy, etc.

Validating a model - Predictive analysis A way validate a model is to compare its output with its actual value on data for which the output we already know. From this kind of analysis, we can compute some statistical attributes from which we can then infer the predictive power of our model by using statistical models. By contrasting the predictive power of different models for different data, we can then decide on what model is the most suitable for a given dataset.

Data compatibility At any given point in time, a predictive model only makes its predictions based on a finite quantity of training data. From this data pool, we can infer a data distribution; a mathematical relation between the data. Assuming we know this distribution, we can compare the input data to it. If the input is too far from the mean or the deviation is too high, it can be very detrimental to the quality of the prediction. Thus, it is important to check if the input data is coherent with the model's data when making a prediction and this independently of the estimated quality of the model. Data compatibility is a criterion to judge the quality of a prediction.

3 Developing ML-based applications

Now that we know of some ways to validate or at least judge the quality of ML applications, we are now interested in exploring tools that would allow us to make real-world applications.

3.1 Getting relevant and "good" data

Predictive models are used to guess an outcome for some data. The data can either be gathered by the user themselves, or by using a preexisting dataset and picking the desired attributes (features). Whatever method we choose, we should keep in mind that the data should be of good quality for the predictive model to work; The data should be clean and of sufficient size, and the features well thought-of.

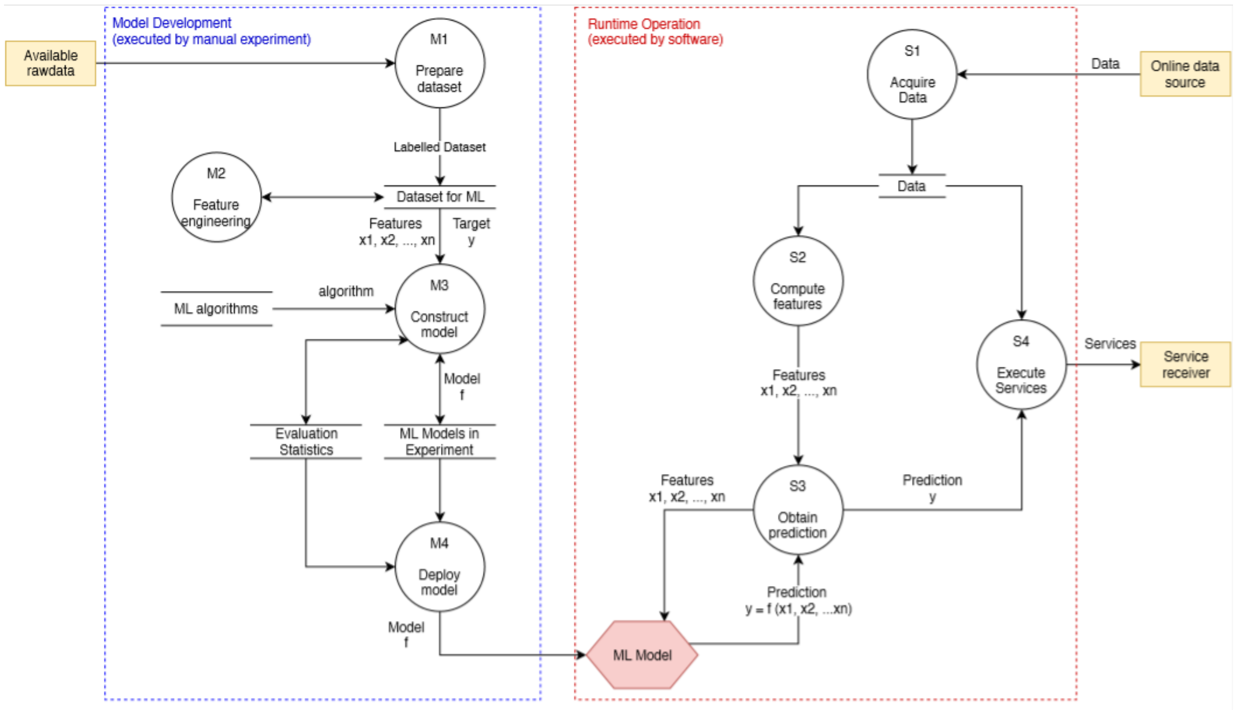


Fig. 2. Data flow of a standard ML-based application with no concern for validation/security issues

3.2 Microsoft Azure Machine Learning Studio

Introduction Azure ML Studio is a web solution to create, test, evaluate and deploy machine-learning-related solutions. It allows to create models ("experiments"), evaluate them and deploy them on the web via web-services.
The user provides their dataset and from then on is able to make operations it (choosing which features to use, normalizing data, splitting it in multiple sets for testing, etc.), apply the core learning algorithm (regression or classification) and then get the resulting model and evaluate it. Different models can be compared for evaluation purposes.

Test data By splitting the input data in more than one blob, it is possible to use some data as "training data" meaning that it won't actually be used to train the model but to evaluate its accuracy.

Deploying and making use of Azure web services Azure offers the possibility to deploy models as web services; This allows authenticated users to query our model(s) for predicting the outcome of a set of input data. Both the requests and responses are formatted in JSON. The service can be called by any language with support for web libraries such as Javascript, Python or C#.
On top of returning predictions, web services can also be used to retrieve metadata or model evaluation results.

3.3 Recognizing the risks and threats

When engineering ML software solutions, it is important to recognize the things that can go wrong beforehand so as to prevent them from having unexpected consequences. The system shouldn't take actions based on predictions considered "bad", or at least adapt the action based on the prediction's quality. In the same fashion, the action undertook by the system should be in accordance with the user's environment.

On top of the predictive aspects and depending on the needs of the system, other computer security concerns should be addressed. Theses threats include, but are not limited to:

- User substitution
- Data tampering
- Data leakage
- Denial of Service (DoS)

It is important to take measures against these threats to guarantee the correct execution of the program.

4 Case study - Wine quality prediction

In the following section, I will present the application I am developing using the methods and concepts I previously described.

4.1 Context

My idea is to create a model which tries to assign a quality indicator (subjective) to a wine bottle depending on the physical (objective) properties of the bottle. By computing the quality of a given bottle, wine resellers may be able to infer the popularity of said bottle without relying on market studies or other non-systematic models which may very well be both less accurate and more expensive. The dataset I used can be found on the UCI website[4]. It contains about 5000 rows (bottles) of 12 columns (physical features + quality).

The main point of this illustrating example is to make a simple yet realistic application, meaning that even if it doesn't reach the quality of industrial software solutions it should still have real-world applications. With that in mind, I thought that being able to systematically infer the quality of wine bottles could very well benefit alcohol re-sellers who wish for example to predetermine which bottles should and shouldn't be considered for buying and in what quantity.

4.2 Building, testing, evaluating and deploying the model using Azure

As stated in the previous section, Azure is used to make the learning backbone of the application. Since the quality indicator is a number, the algorithm used is a regression.
I have done some analysis of 3 different regression algorithms (Linear Regression, Bayesian Linear Regression and Decision Forest Regression) with both normalized and non-normalized values for each algorithm. I decided to settle with Linear Regression with non-normalized data because it ended up being the most accurate model for the training data I used, scoring a mean error of 0.59. The data is split according to the following scheme: 70% of the data is used for the training and the remaining 30% for testing.

After the model has been created, evaluated and tested, it is ready to be deployed as a web service. The user inputs the physical properties of one or more bottles in the same format as the dataset, and finally the model returns the computed quality for each bottle.

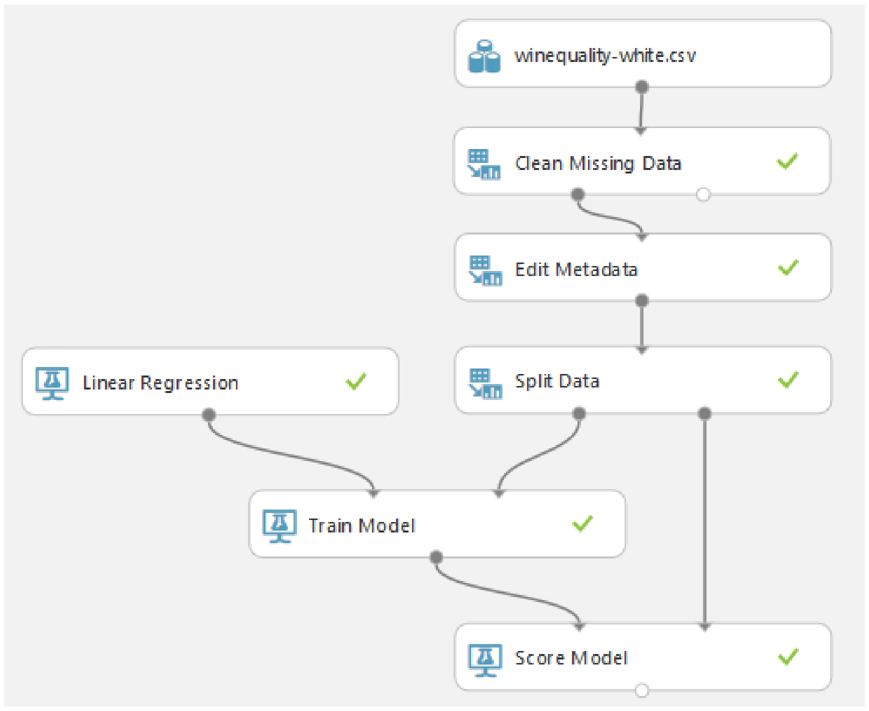


Fig. 3. Wine quality training model

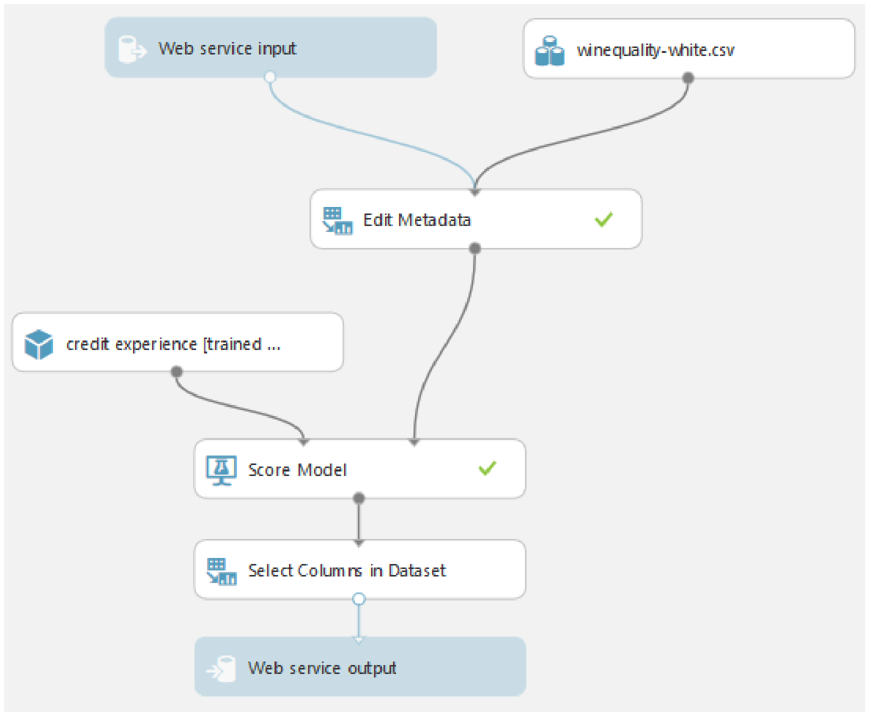
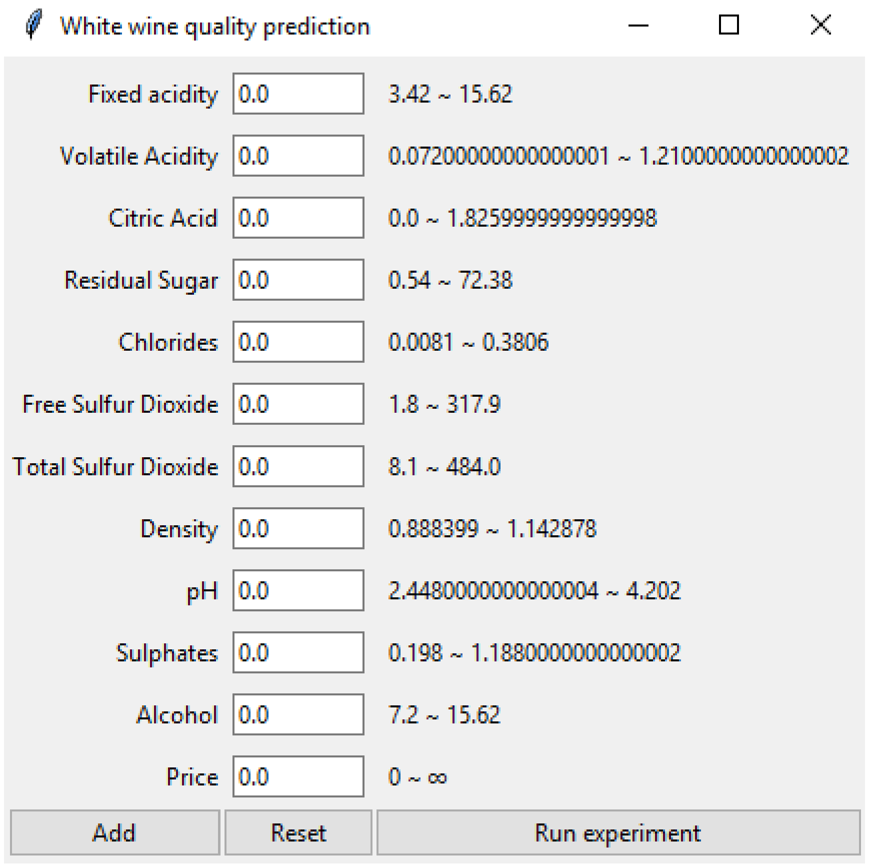


Fig. 4. Wine quality predictive service

Interfacing the ML model with a software application The next steps consists in actually using the model in an application that makes use of it. To that end, I made a Python program with a simple interface which allows the user to enter some wine bottles as well as their price, and the program then queries the web-service to compute the quality for each bottle. The program then returns the bottle with the highest predicted quality and computes the bottle with the best quality/price ratio, which might be the most relevant attribute for the user. Other rules can be defined, such as computing the best quality/price ratio only for bottles with a set minimum required quality.


White wine quality prediction

Fixed acidity	0.0	3.42 ~ 15.62
Volatile Acidity	0.0	0.07200000000000001 ~ 1.2100000000000002
Citric Acid	0.0	0.0 ~ 1.8259999999999998
Residual Sugar	0.0	0.54 ~ 72.38
Chlorides	0.0	0.0081 ~ 0.3806
Free Sulfur Dioxide	0.0	1.8 ~ 317.9
Total Sulfur Dioxide	0.0	8.1 ~ 484.0
Density	0.0	0.888399 ~ 1.142878
pH	0.0	2.4480000000000004 ~ 4.202
Sulphates	0.0	0.198 ~ 1.1880000000000002
Alcohol	0.0	7.2 ~ 15.62
Price	0.0	0 ~ ∞

Fig. 5. Query tool. Interface made with TKinter

4.3 Validating the application

Data coherence The data fed to the model should be in accordance with the model's dataset. A verification of input values' type is made so as to prevent the user from inputting nonsensical data (for example, entering a string when a float is expected). I have decided to limit the interval of possible values for the wine quality to the interval of values already present in the data (plus/minus a 10% difference) so as to make sure that the input data is not too far away from the dataset's. The intervals are retrieved directly from the real version of the dataset uploaded to Azure thanks to a web-service that returns the minimum and maximum value for each attribute when queried.
I have been looking into ways to compute a "data similarity coefficient" to help measure the quality of predictions based on the quality of the input data. A method I envision is to compute the Mahalanobis distance of the input data from the dataset's; the greater this distance, the least accurate the prediction will be. However, computing this distance requires to know the distribution of the dataset, which as of the time I am writing this report I am still unsure on how to go about.

Web-server architecture for security purposes Initially, the user program was given the model's API key to directly query the web-service. However, this solution has many problems; the administrator cannot revoke some user's access without changing the key for everyone, and if the key is compromised by a malicious attacker or user then differentiating between legit and malicious users' requests is impossible.
In order to make up for these problems, I have decided to switch to a server-based architecture. Instead of having the user directly send the queries to the web-service, they have to go through an administrator server which first authenticates the user and sends the request to the web service in their stead. By using python SSH/TLS bindings for communication, we guarantee that impersonation and data For the user to be able to authenticate the server correctly, the later needs to have a valid certificate issued by a certificate authority, or to exchange the certificate with the other in a secure manner (isolated local network with guarantee of no intruders).

Trace analysis As of now, the software doesn't generate any traces to compare properties against. However, I have been thinking about ways to implement this. For example, a trace of all queries done and their result may come in handy to assert that the ML model is deterministic (same input = same output), which it should definitely be. In case an assertion is found to be false, the program can take immediate action so as not to impact the environment.

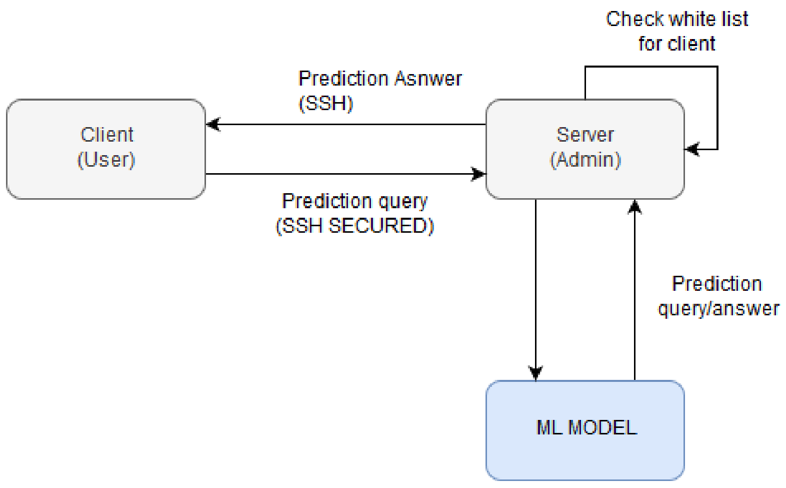


Fig. 6. Client-server based architecture

5 Conclusion

As part of my work so far, I have explored three different ways to validate my application: to make the ML model as reliable as possible, to ensure the security and correctness of the software and finally to adapt the action based on the quality of predictions. In its current state the case-study program doesn't interact with the outside world; it just takes some bottles as input and returns the one it believes is be the best fit for the user and lets the user decide which action(s) are adequate. However if it were to take some actions, then they better do not have any negative influence on the environment. I have yet make a formal, thorough analysis of my model and while I don't think that I have the tools and sufficient knowledge to aspire to do, I do think that recognizing what influences the quality of predictions and and taking it into account in my application is already something. For the remaining month and a half of my internship, I wish to work more on the verification (including risk consideration) of the system as a whole instead of mostly working on the model because as long as we aren't able to prove the correctness of a prediction, I believe that the decisions made should always take the risk into consideration.

6 Acknowledgements

First and foremost, I want to thank Ms. Lydie du Bousquet, Mr. Masahide Nakamura and Mr. Philippe Lalanda who granted us this internship opportunity and for taking the time to instruct us. Thank you to every member of Kobe's CS27 laboratory for their warm welcome and for taking good care of us in Kobe. Last but not least, I want to thank the Grenoble university and the Auvergne-Rhône-Alpes region which helped us finance the internship by paying for the plane tickets.

References

1. du Bousquet, L., Masahide, N.: Improving Testability of Software Systems That Include a Learning Feature. In The: Tenth International Conference on Advances in System Testing and Validation Lifecycle . Nice, France (2018)
2. Blein, Y., Chehida, S., Vega, G., Ledru, Y., du Bousquet, L.: An Environment for the ParTraP Trace Property Language (Tool Demonstration). Runtime Verification, pp. 437–446. Springer International Publishing (2018)
3. Yoann Blein - ParTraP : Un langage pour la spécification et vérification à l'exécution de propriétés paramétriques, https://www.liglab.fr/en/events/thesis-defenses/yoann-blein-partrap-un-langage-pour-la-specification-et-verification-a. Last accessed 11 Jun 2019
4. UCI Wine Quality dataset, https://archive.ics.uci.edu/ml/datasets/Wine+Quality. Last accessed 2 Jun 2019
5. Internship github repository, https://github.com/muretti0114/UGA2019. Last accessed 13 Jun 2019