

Bartcus_Marius_2_outil_suivi_et_analyse_methodologie_pilotage_performance_022023

The report consists of three sections:

- To begin with, I demonstrate how the LUIS model is evaluated to understand its performance. Our approach is to compare the actual entities with those evaluated and compute an accuracy score based on these comparisons.
- Secondly, I will demonstrate how Azure Application Insights analyzes chatbot activity.
- Lastly, I describe the methodology for monitoring model performance in production by illustrating the current and target architectures.

Model Evaluation

It is necessary to evaluate the model in order to understand its performance. The LUIS model must be evaluated both in terms of its accuracy and performance as it pertains to understanding user intents and extracting relevant information from the input of the user.

The test dataset is used to evaluate the model. A comparison is made between the predicted and actual entities. If the predicted entity matches the real one, a value of 1 is added to the true value. If part of the predicted entity is contained in the true entity, the correct value is increased by 0.5. Our accuracy score is calculated as follows:

```
(correct_entity / len(true_entities)) * 100
```

Based on the test data, we obtained a precision of 59.71% between the predicted model and the actual liberalization of the original data. Figure 1 shows each utterance was evaluated. Several utterances were evaluated with 100% precision. Many of them predict nothing, while others have middle precision levels. We can examine these three types of utterances by observing them.

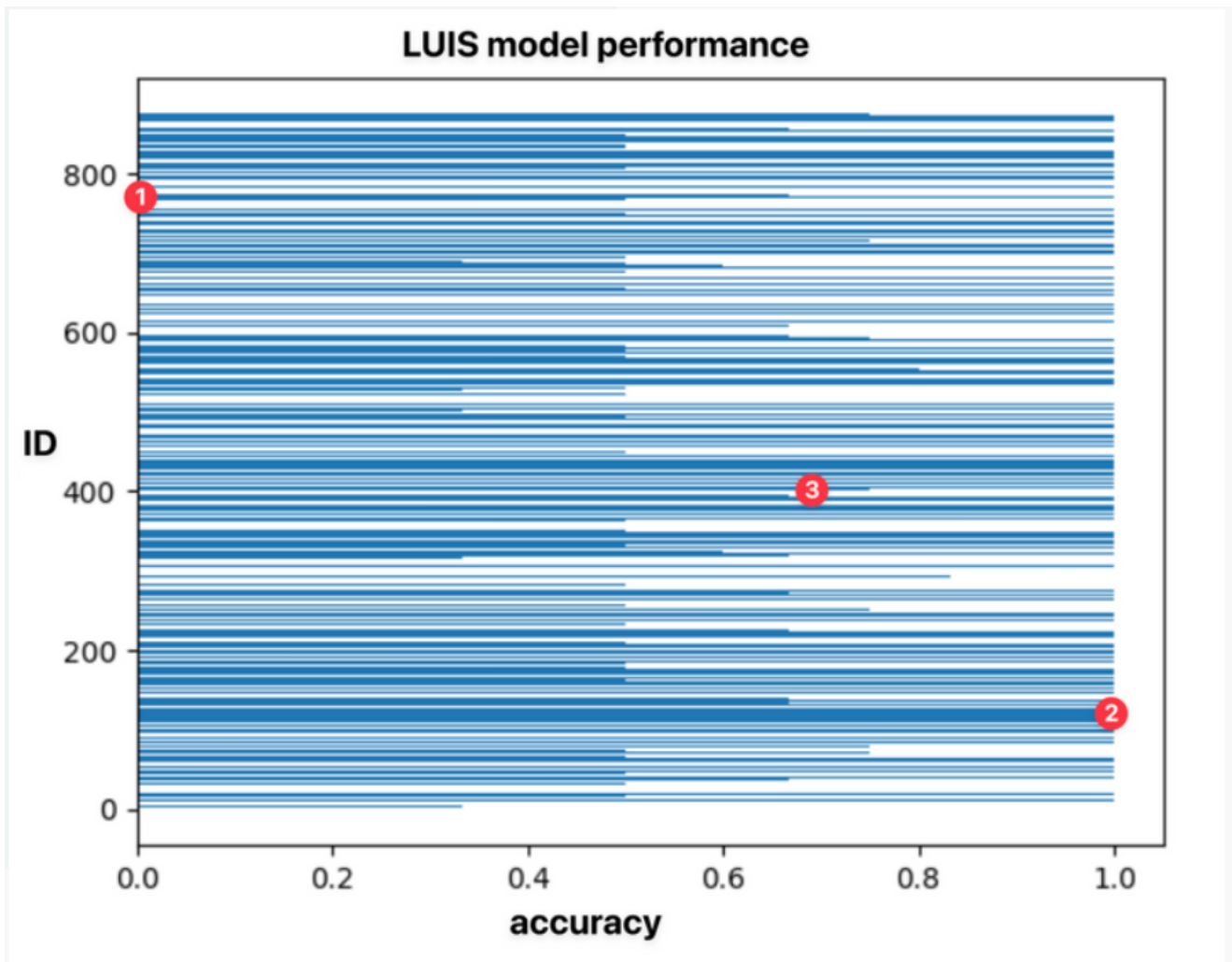


Figure 1: Evaluating model.

First, let us examine the case where entities can be predicted with 100% accuracy. Here, our test data looks like this:

```
{ 'text': 'Can anything help resurrect my career at this point??? Is there  
any sadder rung in society than that of the failed writer\n0k you know  
what, maybe you can help\nGet me to Campinas to San Juan please\nMe, my  
wife, and my son',  
  
'intent_name': 'BookFlight',  
  
'entity_labels': [{ 'entity_name': 'or_city',  
  
'start_char_index': 169,  
  
'end_char_index': 177},  
  
{ 'entity_name': 'dst_city', 'start_char_index': 181, 'end_char_index':  
189}]]}
```

The true entities are:

```
{'or_city': 'Campinas', 'dst_city': 'San Juan'}
```

while the predicted entities are:

```
{'or_city': 'Campinas',  
  
'geographyV2': {'value': 'Campinas', 'type': 'city'},  
  
'dst_city': 'San Juan'}
```

We see that the `or_city` and `dst_city` are predicted by the LUIS model as expected.

Let us now examine the case where we have 66.66% accuracy.

```
{'text': 'Can I take my wife and son from Madrid to Santos for 4800??',  
  
'intent_name': 'BookFlight',  
  
'entity_labels': [{'entity_name': 'or_city',  
  
'start_char_index': 32,  
  
'end_char_index': 38},  
  
{'entity_name': 'dst_city', 'start_char_index': 42, 'end_char_index': 48},  
  
{'entity_name': 'budget', 'start_char_index': 53, 'end_char_index': 57}]}
```

The true entities are:

```
{'or_city': 'Madrid', 'dst_city': 'Santos', 'budget': '4800'}
```

We obtain the predicted entities:

```
{'or_city': 'Madrid',  
  
'dst_city': 'Santos',  
  
'geographyV2': {'value': 'Santos', 'type': 'city'},  
  
'number': 4800,  
  
'budget': '4800??'}
```

The budget entity is the source of this problem. Observe that the predicted entity took into consideration the questions at the end of the number. This is because there were no spaces here, and cleaning the dataset could resolve this issue. However, for this project, I do not consider this.

Finally, what happens to the utterances that are examined with 0% accuracy? The same as for the 66% accuracy case. We find entities that contain some characters at the end, which explains that the predicted entity takes 0% of accuracy.

Analysis of chatbot activity in production using Azure application insight

We use Application Insights to control the model's performance in production. This will permit us to answer the question of "how many customers used the application. It will give us availability and performance monitoring. Go to the Azure portal and create the application Insights as in Figure 1.



Application Insights [Add to Favorites](#)

Microsoft | Azure Service

★ 4.3 (509 ratings)

Plan

Application Insights

Create

How are customers using my app/website?

Application Insights answers these questions and many more. With Application Insights, you have full observability into your application across all components and dependencies of your complex distributed architecture, as well as:

- Availability and performance monitoring
- Deep code diagnostics with profiler and debugger
- Users and usage insights
- Out of the box performance and failures triage with end-to-end transaction diagnostics
- Seamless integration with Microsoft Azure and Visual Studio
- Supports ASP.NET, .NET Core, Java, Node.js applications and websites on Azure or on-premises

Figure1: Creating application Insights.

Check the resource mode **Classic** and click on **Renew + create** as in Figure 2. Then check the summary and click **Create**.

Application Insights

Monitor web app performance and usage

Basics Tags Review + create

Create an Application Insights resource to monitor your live web application. With Application Insights, you have full observability into your application across all components and dependencies of your complex distributed architecture. It includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. It's designed to help you continuously improve performance and usability. It works for apps on a wide variety of platforms including .NET, Node.js and Java EE, hosted on-premises, hybrid, or any public cloud. [Learn More](#)

PROJECT DETAILS

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ


Resource Group * ⓘ [Create new](#)

INSTANCE DETAILS

Name * ⓘ

Region * ⓘ

Resource Mode * ⓘ ☒ Classic ☐ Workspace-based

 Classic Application Insights is deprecated and will be retired in February 2024. Please consider using Workspace-based Application Insights. [Learn more about Workspace-based Application Insights resources.](#)

[Review + create](#)

[« Previous](#)

[Next : Tags >](#)

Figure 2: Creation of Application Insights Classic.

Once we created our Application Insights resources, we can use the instrumentation key to configure our application insights sdk. We can copy it to the environment variables (.env file) to use it in our bot that will be created.

We add application insights to monitor the bot's performance and monitor the problematic interactions (experiences) between the chatbot and the user. Therefore, we use the final step to log in the application insights with AzureLogHandler. Furthermore, we track two details about a user's experience. The first thing to check is

if the flight was satisfied with the bot's proposals and booked. And the second is if the customer was not satisfied with the bot's proposals. If the flight was booked, we log it with the level INFO. A customer who is not satisfied is logged with the level ERROR.

```
async def final_step(self, step_context: WaterfallStepContext) ->
DialogTurnResult:
    """Complete the interaction and end the dialog."""
    booking_details = step_context.options

    if step_context.result:
        self.logger.setLevel(logging.INFO)
        self.logger.info('The flight is booked and the customer is
satisfied.')

        return await step_context.end_dialog(booking_details)

    prop = {'custom_dimensions': booking_details.__dict__}

    self.logger.setLevel(logging.ERROR)
    self.logger.error('The customer was not satisfied about the bots
proposals', extra=prop)

    return await step_context.end_dialog()
```

In addition, we trigger an alert if the user does not accept the bot's proposal three times within five minutes. The alert is created under Application Insights, Alerts. We click **Create** followed by **Create rule**, and set up our alert. Here is an example of an alert I create.

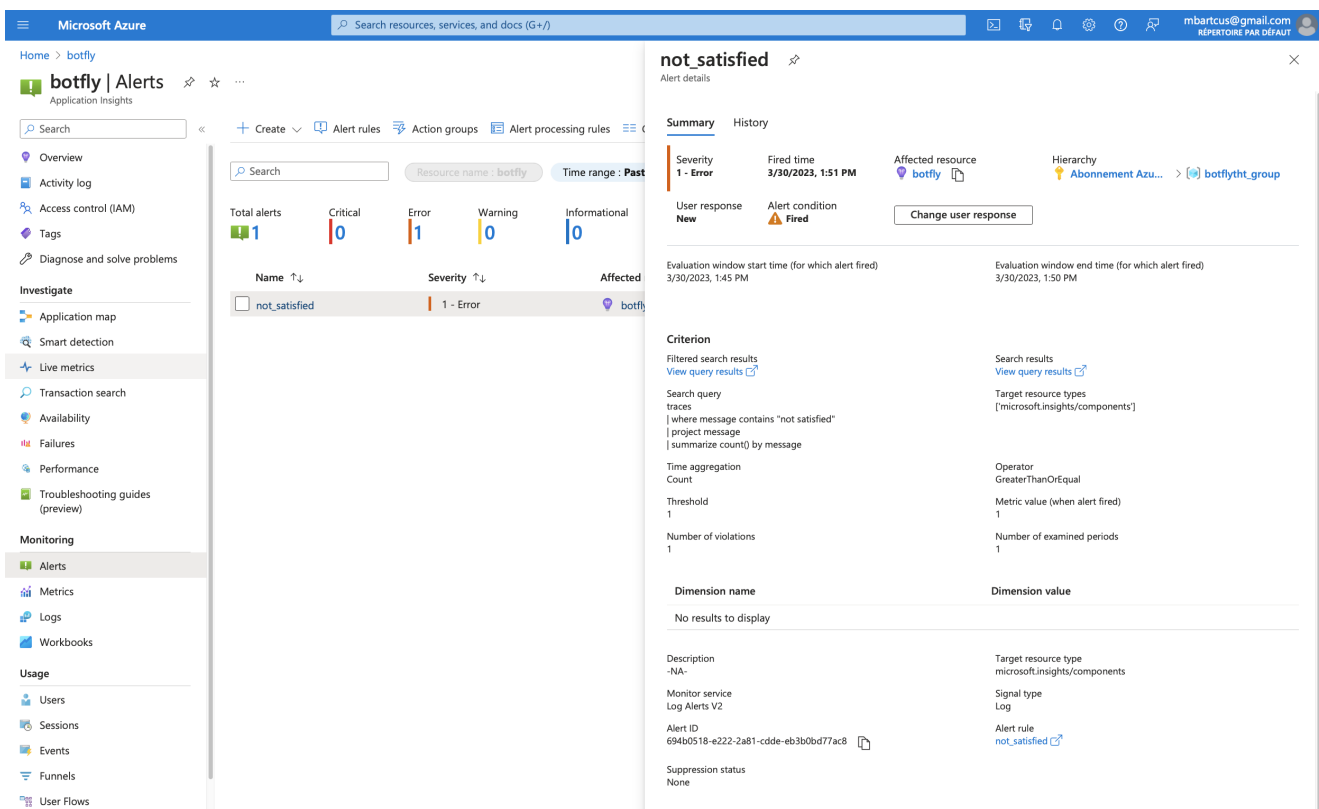


Figure 3: Viewing the Alert.

As can be seen, the ERROR level log was triggered three times in five minutes. Therefore in the Error section, the alert was created. It is also possible to view the query that initiated the alert. We look for "not satisfied" in a trace message. The alert will be created if an item appears three times.

In future work, we can use application insights to:

- monitor the number of requests that are made to our bot
- monitor the number of errors that are made by our bot (ex: the user does not enter the correct date format)
- monitor the number of users that are using our bot
- monitor the number of times the user does not finalize the booking process
- etc.

Methodology for monitoring the performance of the model in production

The purpose of this section is to demonstrate the methodology used to monitor the performance of the model in production for this project. Figure 4 depicts the current architecture for MVP, which is the first version of the application. This type of architecture can not monitor the performance of the model in production. The disadvantage is the fact that we can not update our model regarding some new data. It should not be considered definitive. Our future work will focus on developing a

target architecture that will enable us to monitor and update our model. As a result, the chatbot will be more efficient.

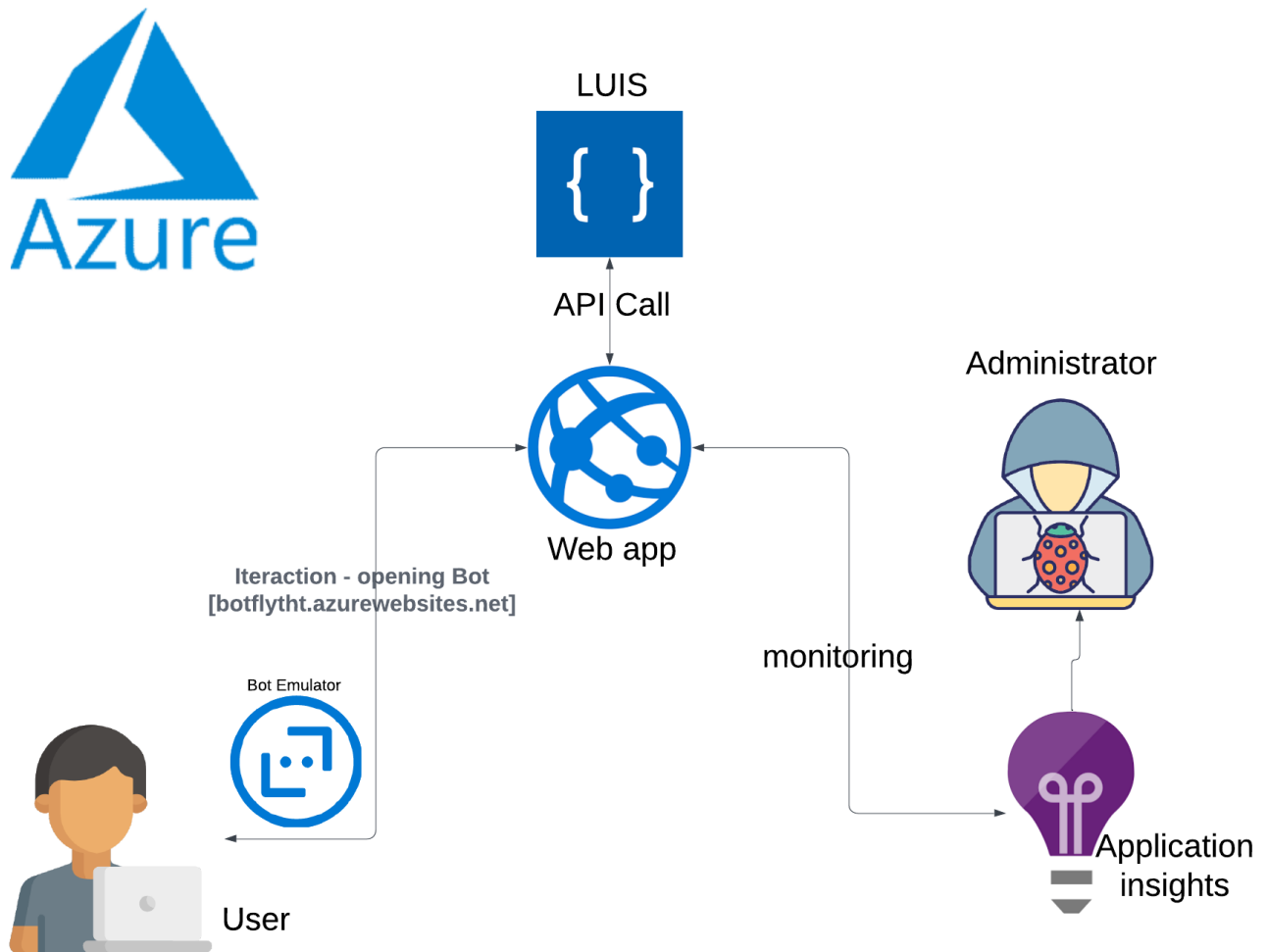


Figure 4: Current Architecture.

Figure 5 shows the target architecture that improves the methodology for monitoring the model's performance in production. The future version of the application will incorporate an architecture that allows us to collect the conversations between the bot and the user. We would add them to the data. Having the latest data, already formatted for the LUIS model, we could learn the updated model, using Azure Functions, in a month. If the updated model gives us better performance (regarding the model evaluation in Figure 1) we would update the model using an Azure Function.

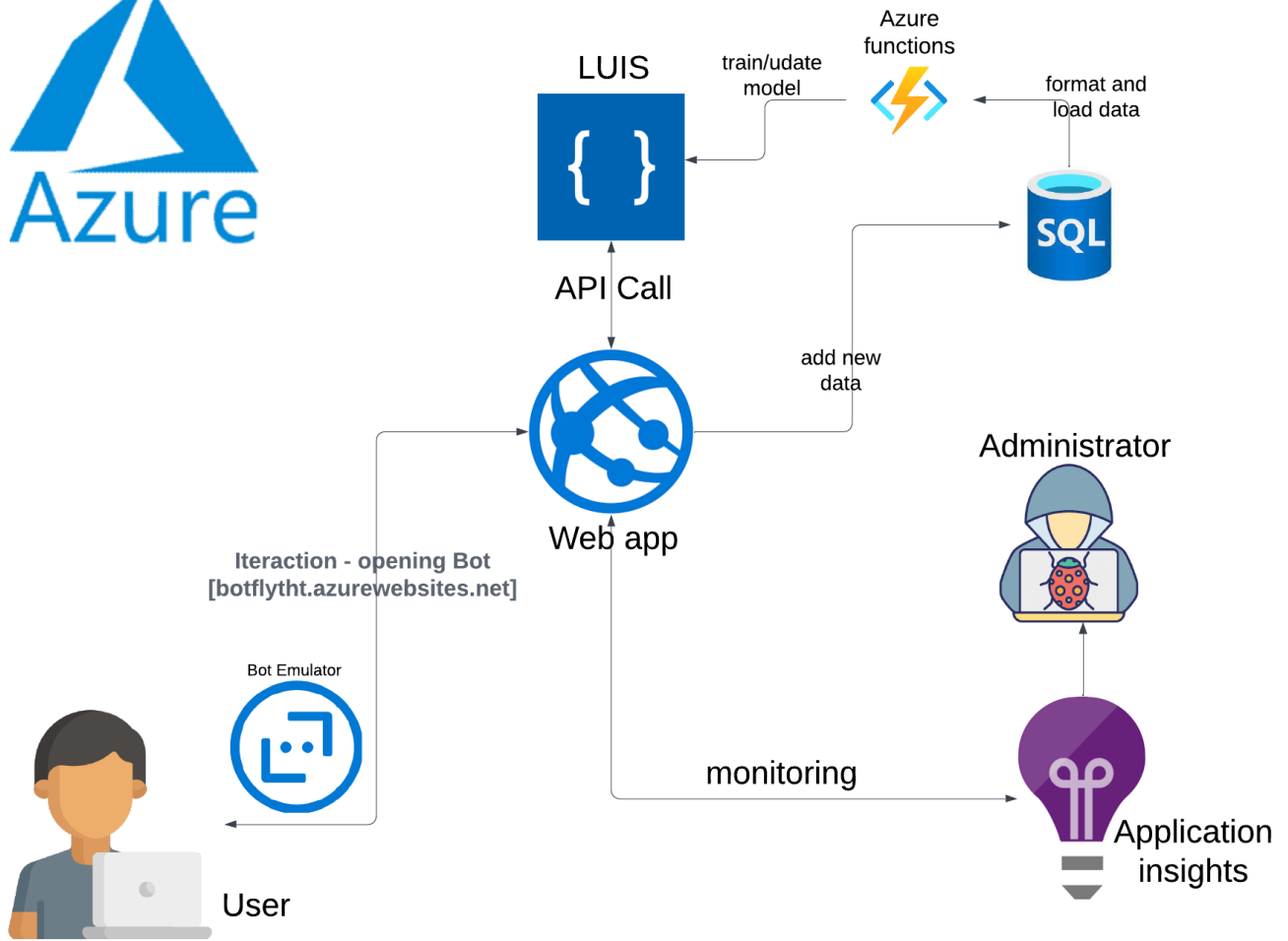


Figure 5: Target Architecture.