Enable Natural Language Interaction: LUIS to your rescue

The technological landscape has changed quite radically in recent years. Computing capabilities moved from PCs to smartphones and wearables while adding even more power to these devices. This advancement in technology has also changed the way we interact with our devices. Physical keyboards were replaced with software implementations and the input methods changed from stylus to mere tap with fingers. It was only a matter of time before we started looking for even more effortless ways to interact with our computing devices.

Speech is how we interact with each other and now we are at the verge of using speech to also interact with all our smart devices. The recent launch of Bot Framework and Cognitive Services at //BUILD are a step towards that vision. Among these amazing cognitive services, 'LUIS - Language Understanding Intelligence Service' provides us with the capabilities to understand the natural language queries and return actionable information that we can wire up in our code to improve the user experience.

In this article, we will explore the capabilities of LUIS and look at different ways we can use it in our apps and services.

How does LUIS work?

LUIS is built on the interactive machine learning and language understanding research from Microsoft Research group. The book "Machine Learning" by Tom Mitchell in 1997 defines machine learning as –

> *"A computer program is said to learn to perform a task T from experience E, if its performance at task T, as measured by a performance metric P, improves with experience E over time."*

Like any other machine learning paradigm, LUIS is an iteration of the above-mentioned concept. It uses a language model and set of training examples to parse spoken language and return only the interesting parts which we, as developers, can use to delight our users.

With LUIS, apart from using your own purpose-specific language model, you can also leverage the same pre-existing and pre-built language models that are used by Bing and Cortana.

LUIS has a very specific use case – you can leverage LUIS anywhere you have a need to allow the users to interact with your apps using speech. Most digital assistants, voice-enabled apps/devices and Bots fall in to this category but you are free to use your imagination.

Where can I use LUIS?

Using LUIS with our apps and services requires initial setup. The homework we need to complete consists of understanding the scenario and anticipating the interaction that will take place between the apps and the users. Understanding and anticipating the interaction will help us build the language model to use with LUIS and to come up with the basic natural utterance to train LUIS parse them.

Language models

Language models are specific to your LUIS applications. They form the basis of understanding what the users mean when talking to our apps and services. There are two core parts of a language model – *Intents* and *Entities*. LUIS application uses the intents and entities to process the natural language queries and derive the intention and the topics of interest to the users with help from the training examples also called utterances. LUIS applications always contain a default intent called *None*. This intent gets mapped to all the utterance which couldn't be mapped to any other intents. In the context of an app, intents are actions that the users intend to perform while the entities get filtered to the topics that your apps and services are designed to handle.

An example to understand this would be imagine a shopping app and the following model –

```
"intents": [
  {
    "name": "ShowItems"
  }
  {
    "name": "BuyItems"
  }
],
"entities": [
  {
    "name": "Item"
  }
]
```

Majority of time spent in a shopping app would be to go through the items on sale and when someone says – "*Show me red scarves*" – the model mentioned above will map this utterance to *ShowItems* as the intent and "*red scarves*" to the entity *Item*. At the same time, you can map an utterance to the *BuyItems* intent and thus initiate the checkout process when someone says – "*I would like to pay now*".

Intents

LUIS ***intents*** also support action binding which allows you to bind parameters to your intents. Actionable intents fire only when these parameters are provided as well. In particular, when using action binding with Bots, LUIS can query the parameters from the users interactively.

Based on the examples and active learning, LUIS starts detecting the intents in the queries posted to it. However, since the language queries are tricky for the computer applications, LUIS also scores them between 0 to 1 to denote its confidence – higher scope denotes higher confidence.

Entities

LUIS ***Entities***, as explained above, are the topics of interest to your users. If you are building a news app, entities will map to the news topics and in case of weather apps, they would map to locations in their very basic iterations.

LUIS's active learning also starts showing up when you add a new utterance and you can see the appropriate entities color-coded to show the mappings visually.

Entities can have child elements and each of them can be independently mapped to individual parts of the utterances. We also have support for composite entities which are a collection of characteristics which collectively build up to a single entity.

For better understanding, an entity called ***Vehicles*** can have child entities called ***Cars*** and ***SUVs*** (and more). This relationship can help you map multiple different entities in to a larger category. In case of ***Composite Entities***, the individual parts would denote one single with various properties of it. Most appropriate example for a composite entity called **Car** will be ***2016 Black Ford Mustang*** where it is made up of ***Year, Color, Make*** and ***Model*** information.

Pre-built Entities

Similar to data types in any programming language, LUIS service includes a set of entities for a number of common entities so you don't have to go out and think about every possible term that your users may throw at you. Some examples include most common variations of date, time, numbers and geographical entities. You can include the pre-built entities in your application and use them in your labelling activities. Keep in mind that the behavior of pre-built entities cannot be changed.

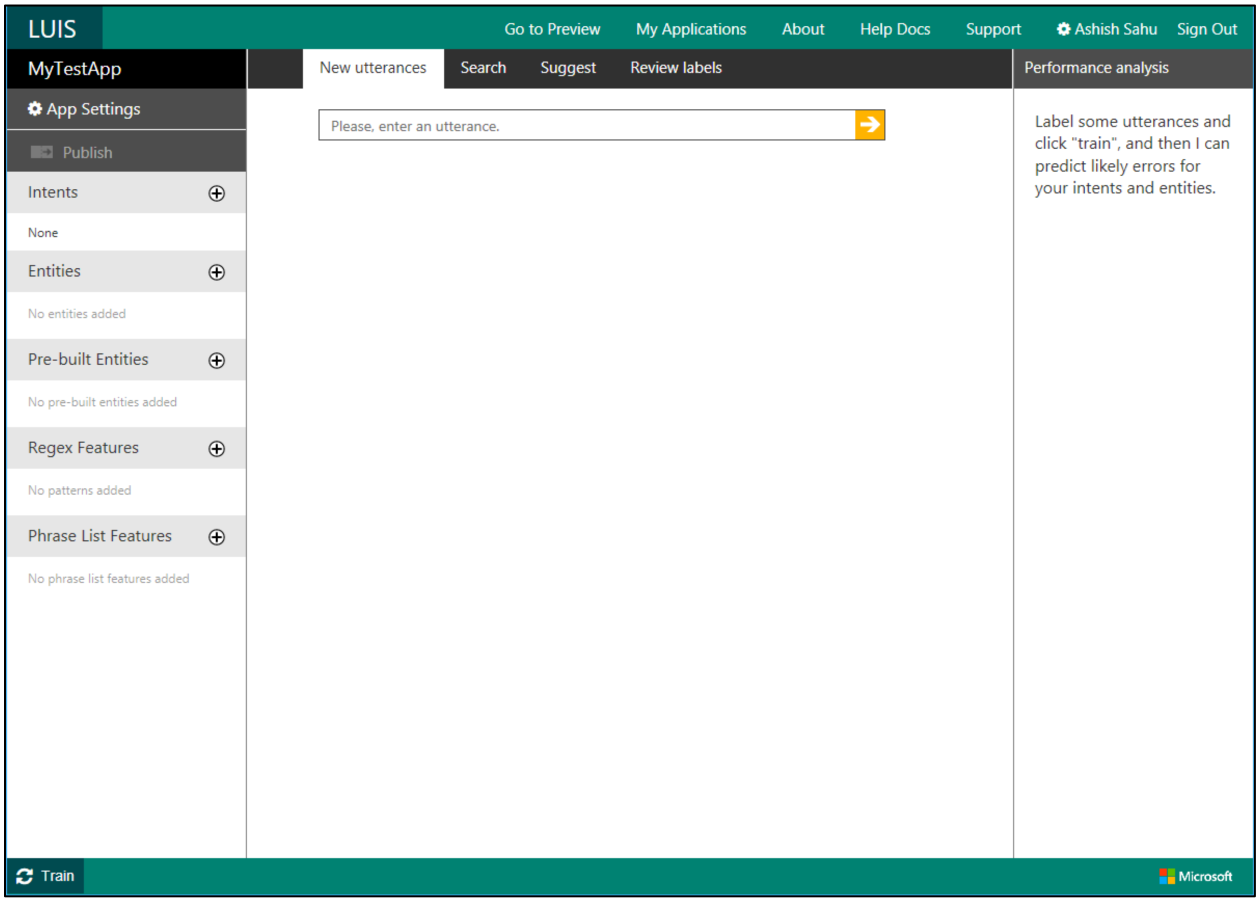
An exhaustive list of pre-built entities can be found in the LUIS documentation.

While is possible to add a number of intents and entities (and pre-built entities) in your model, word of caution is to keep it simple and precise. You can start with the most obvious utterances and add more of them to make it more natural for your users but keep in mind that thinking ahead of the experience you want to build goes a long way in enhancing the user experience and evolving them further. If you don't plan ahead and change intents or entities in your models – you will have to label all the utterance and train your model all over again.

Let's go build something!

It's time to build something and take a ride with LUIS. For this article, we are going to look at an inventory application. We will build a language model using the intents and entities, train the model, and then use this in a Bot powered by the Bot Framework and a UWP app using the REST endpoint that LUIS exposes for us to use all its capabilities.

To keep it simple, we are going to deal with an inventory of clothes. To get started, log on to the LUIS portal at http://www.luis.ai and create a new application. ***Figure 1*** represents the screen you will be greeted with once the LUIS application has been created.



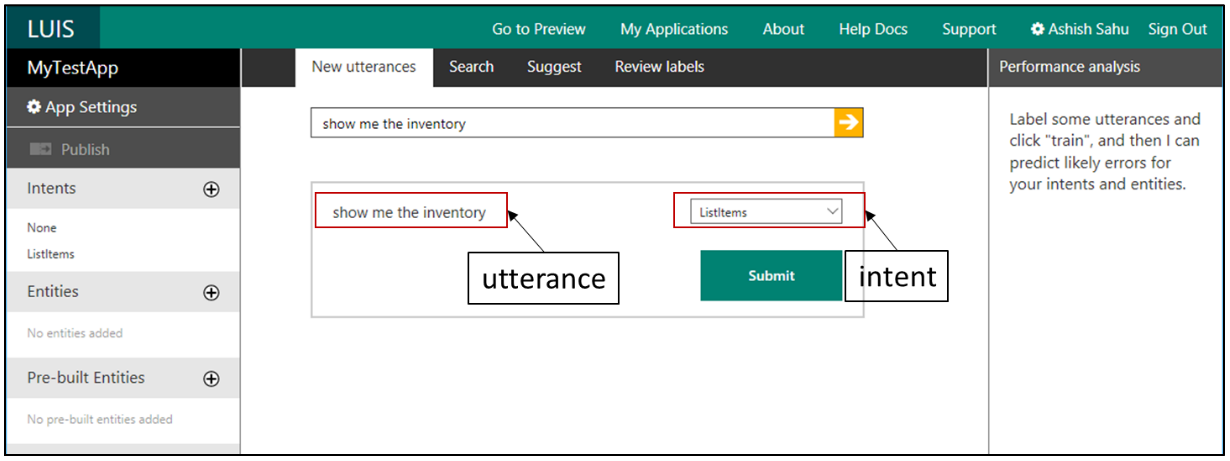
Label some utterances and click "train", and then I can predict likely errors for your intents and entities.

Figure 1

This being an inventory application, we will be using it to track inventory of stocks and for that purpose, the first intent that we are going to add is – *ListItems*. We will map this intent to all the utterances where the user's intent is to query the inventory as a whole or for an item.

When you are creating an intent, the LUIS application will for an example utterance. This utterance is going to be the first natural language query to trigger this intent. Click on the "+" button next to "Intents" and add the name of the intent as "*ListItems*". We will keep the example utterance simple – "*show me the inventory*".

Saving this intent takes you to the "*new utterance*" screen and *Figure 2* shows the example utterance along with the *ListItems* intent mapped to it with in the drop-down menu next to it.



Label some utterances and click "train", and then I can predict likely errors for your intents and entities.

Figure 2

Click on the Submit button to add this utterance to your LUIS application. Before LUIS can start working its magic for us, we must add more such utterance to help LUIS understand the intents more accurately. Keep in mind that the utterances must project the same intent as the one in the screenshot above but at the same time, they should be something that users will say naturally when asking for stocks – "*show me the stocks*" comes to my mind.

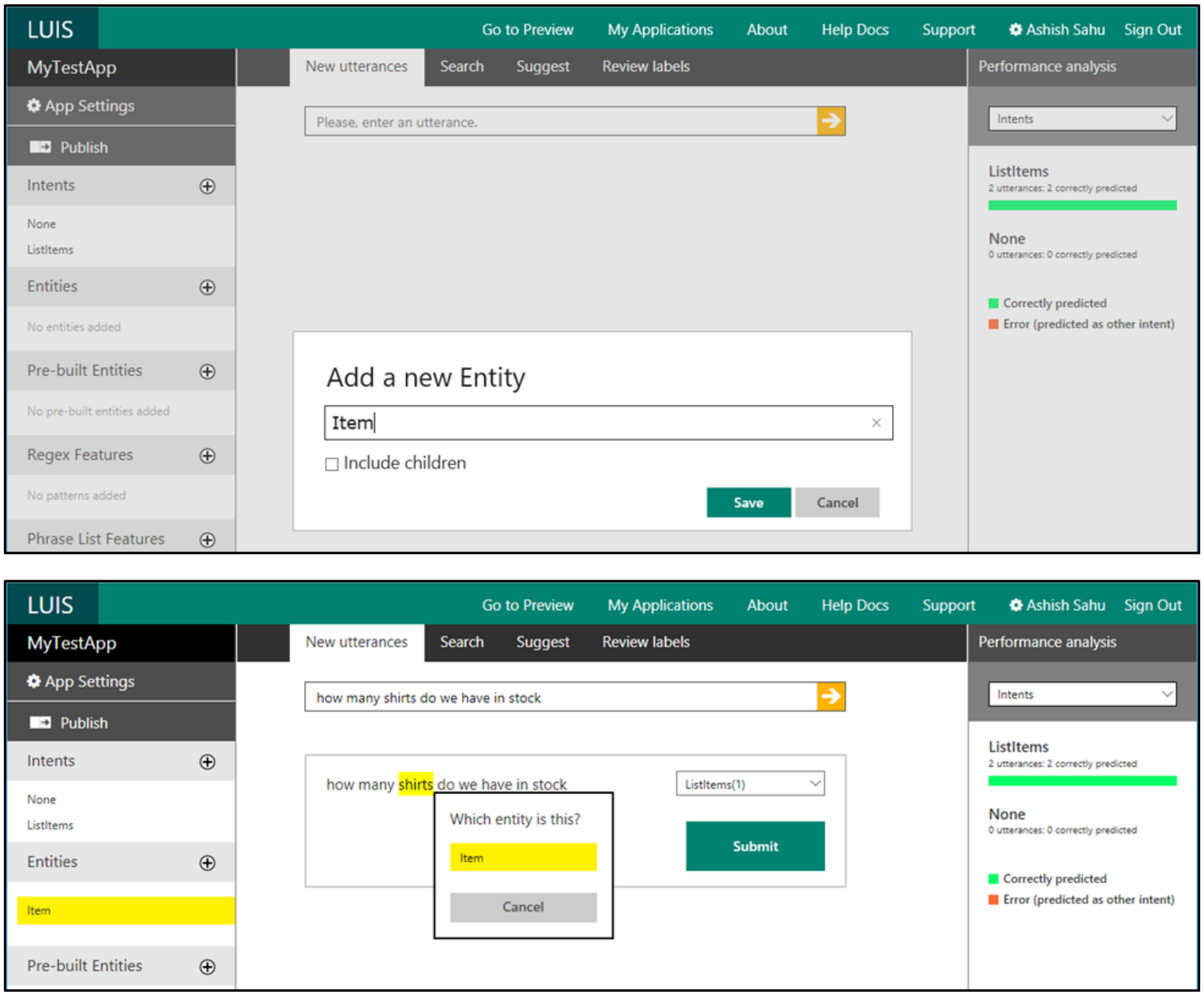
Now that we added two utterances to our application, we can click on the *Train* button the lower left corner to see if the LUIS application has enough information to understand when to trigger the *ListItems* intent. The framework triggers the training periodically on its own as well. Training your model after adding a few examples can help you identify any problem with the model early and take corrective actions. Since LUIS framework also features active learning, you will also benefit from these training as the example utterance will be scored automatically for you as you go along adding them.

Moving forward with our application, it is also very natural for ask about the inventory of certain items we are stocking so let's also think about the examples like – "***Show me the stocks of red scarves***" and "***how many shirts do we have in stock***".

However, these queries are different from the ones we have added so far. These queries contain these terms – ***red scarves*** and ***shirts***. This means we need more than our intent – ***ListItems*** – to return the right results back to our users – we need an entity called ***Item*** which we will map to these terms to add more intelligence in our language model.

We can go ahead and add these utterances to our LUIS application and label the entities later but in this case, we will the add entity first and then the utterances. Click on the "+" button next to ***Entities*** and name this entity ***Item***.

Now, we can add those queries mentioned earlier and label them with the intent and entity at the same time. To do that, just add your utterance and if the intent has not already been mapped with a score, select the relevant intent, and then click on the term ***shirts*** to map it with the ***Item*** entity. Refer to the ***Figure 3*** to understand the process.



Add a new Entity

Item

☐ Include children

how many shirts do we have in stock

Which entity is this?

Item

Figure 3

Select *Item* from the list to label it an item. Add the other example mentioned above now - "*show me the stocks of red scarves*" and instead of mapping just *scarves*, select *red* and *scarves* both as the entity Item. Just a note – our favorite browser Edge doesn't allow you to select multiple words in the LUIS portal. Use any other browser of your choice to do this.

Please also note that the term *red scarves* also falls in the category of *Composite Entities* since they denote one single entity together – *scarves* which have *red* in them. As explained above, *Composite Entities* are made up of multiple parts but represent one unit of object such as *black shoes* and *2016 Ford Mustang*. However, for the sake of simplicity, we are going to treat them as a single entity.

Train the model again and see if the active learning in LUIS kicks in. Let's try adding an utterance such as "*how many wallets do we have in stock*" or "*show me the stocks of trousers*".

You may find the result interesting - notice that the term ***wallets*** gets mapped to ***Item*** entity but ***trousers*** don't. Don't panic – it just means that LUIS needs a few more examples to make sense of utterances that follow the same pattern. To do that, let's map ***trousers*** to ***Item*** entity and train our model one more time.

To test this, try adding "***show me the stocks of red shirts***" or "***show me the stocks of pants***" and verify that ***red shirts*** and ***pants*** get mapped to the right intents and entities. I hope your mileage doesn't vary from mine so far.

Using the ***Suggest*** section in the portal, you can also get the suggestions from the Cortana logs for individual intents and entities.

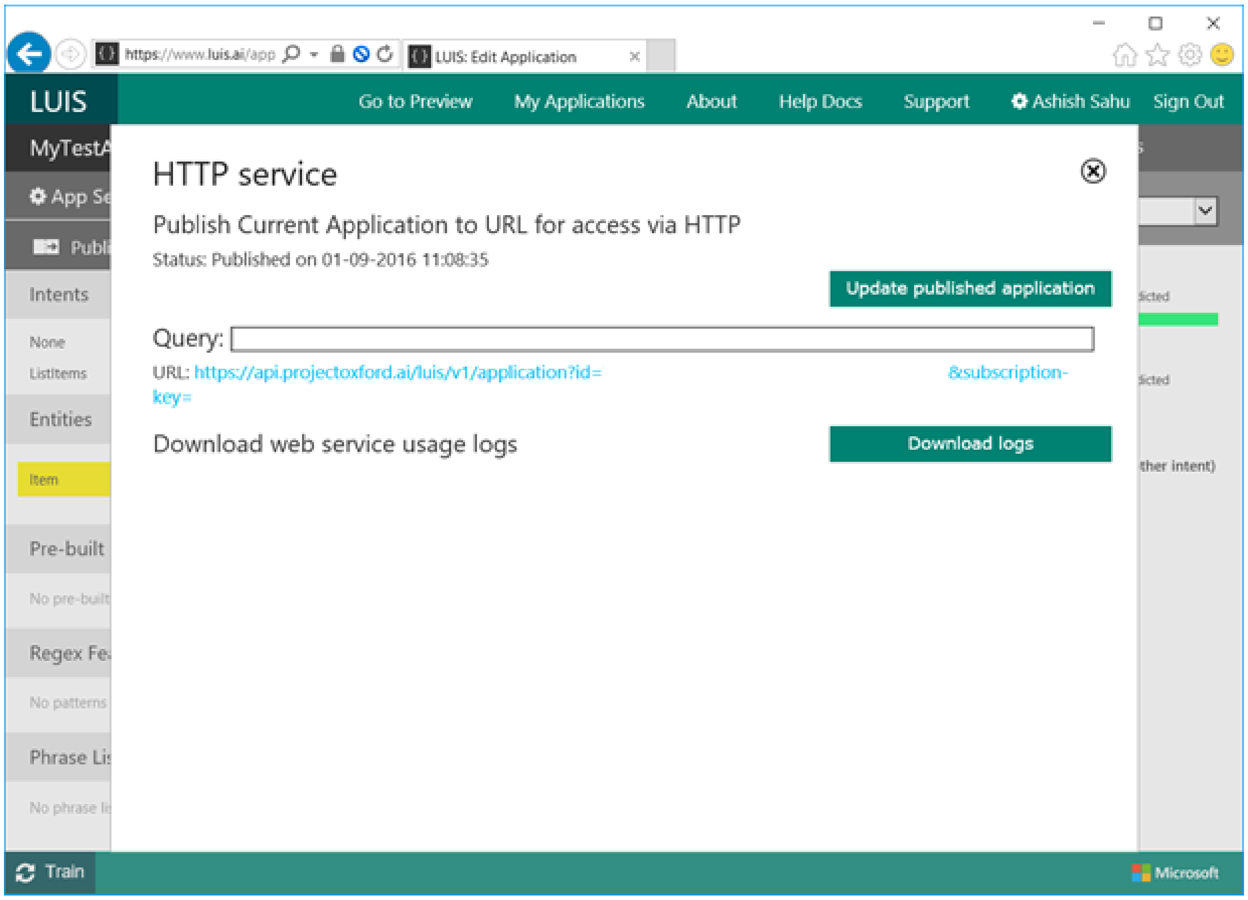
Once our intents and entities are getting mapped correctly, we can now move on to the next phases of our journey on LUIS.

Using LUIS with real apps

This LUIS application is not useful for our users now – we need to connect to this application from our apps and services and since LUIS application are exposed via REST endpoints and the responses are returned in JSON format, you can use LUIS services from any platform or programming language that can connect using HTTPS protocol and parse JSON responses.

Note: The LUIS portal also exposes the export functionality from the "My Application" portion which exports your LUIS application as a JSON document to make changes offline and import it back. In combination with the LUIS APIs and the C# SDK, you can integrate LUIS in your DevOps processes as well.

We also need to publish our LUIS app before we can start calling it from our apps and it's as simple as it gets – just click on the *Publish* button as shown in *Figure 4* and click again on the *Publish web service* button.



HTTP service

Publish Current Application to URL for access via HTTP

Status: Published on 01-09-2016 11:08:35

Query:

URL: https://api.projectoxford.ai/luis/v1/application?id= &subscription-key=

Download web service usage logs

Figure 4

Notice that the REST endpoint URI also includes your LUIS application ID and the subscription key. Protect this information as much as you would any other credentials and it can lead to disruption of the service and financial impact.

Once the application has been published, you should be able to test it by typing any other example in the Query input box and test the accuracy of your model. Let's try that – enter "*how many ties do we have in the stock*" and press Enter on your keyboard.

This will open a new browser window and you should get a response back in the JSON format as shows in the *Figure 5* –

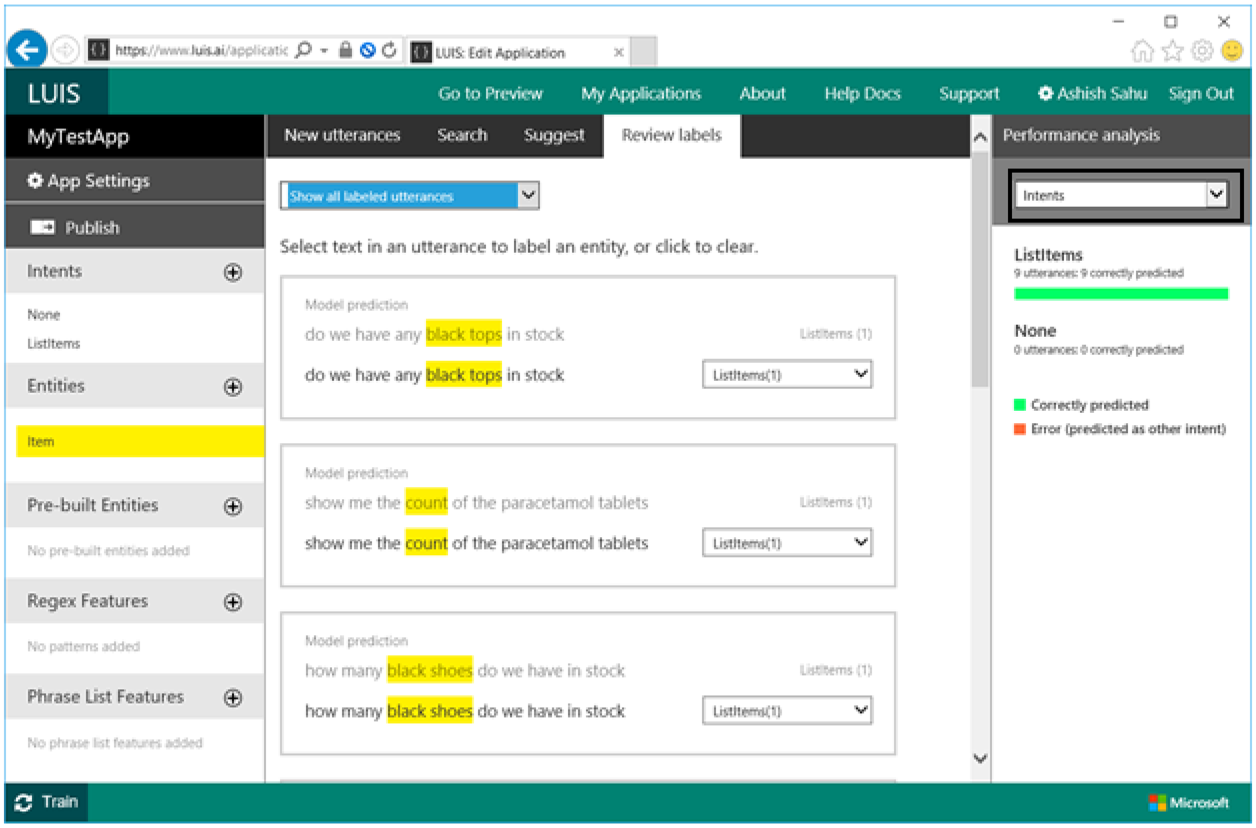
```json
{
  "query": "how many ties do we have in the stock",
  "intents": [
    {
      "intent": "ListItems",
      "score": 0.9999995
    },
    {
      "intent": "None",
      "score": 0.0582637675
    }
  ],
  "entities": [
    {
      "entity": "ties",
      "type": "Item",
      "startIndex": 9,
      "endIndex": 12,
      "score": 0.903107
    }
  ]
}
```

Figure 5

The response includes the query string passed to the LUIS application along with the intents and entities detected in the query. Also included is the individual scoring information for each of them. These scores are important since they are direct indicators of how your language model and the training is performing. As you go on adding more utterance and make any changes to your model, this dialog box also provides you with an option to publish your updates. Updating your LUIS application after every training session is important since it will keep using the older training model and the response from the HTTP endpoint will defer from your expectations.

Analyzing performance of Language Model

Adding too many variations of the language can result in errors and may force you to change your language model. To address these issues, the LUIS portal features a *Performance Analysis* section. You can use this section to understand your LUIS application is performing when it comes to detecting the intents and entities. You can get color-coded performance overview of all of your intents and entities in this section. Depending on the training, examples and language model used, your LUIS application may also run in to issues where it is unable to map intents or entities correctly. There may also be cases where adding multiple types of utterance confuses the LUIS service. These issues can be easily tracked down with the performance drill-down using the Performance analysis. *Figure 6* and *7* below show the performance for our model for all of the intents in our language model as well as the *ListItems* intent in detail. The drop-down menu also allows drill down analysis to individual intent and entities.



Show all labeled utterances

Select text in an utterance to label an entity, or click to clear.

Model prediction
do we have any black tops in stock — ListItems (1)
do we have any black tops in stock — ListItems(1)

Model prediction
show me the count of the paracetamol tablets — ListItems (1)
show me the count of the paracetamol tablets — ListItems(1)

Model prediction
how many black shoes do we have in stock — ListItems (1)
how many black shoes do we have in stock — ListItems(1)

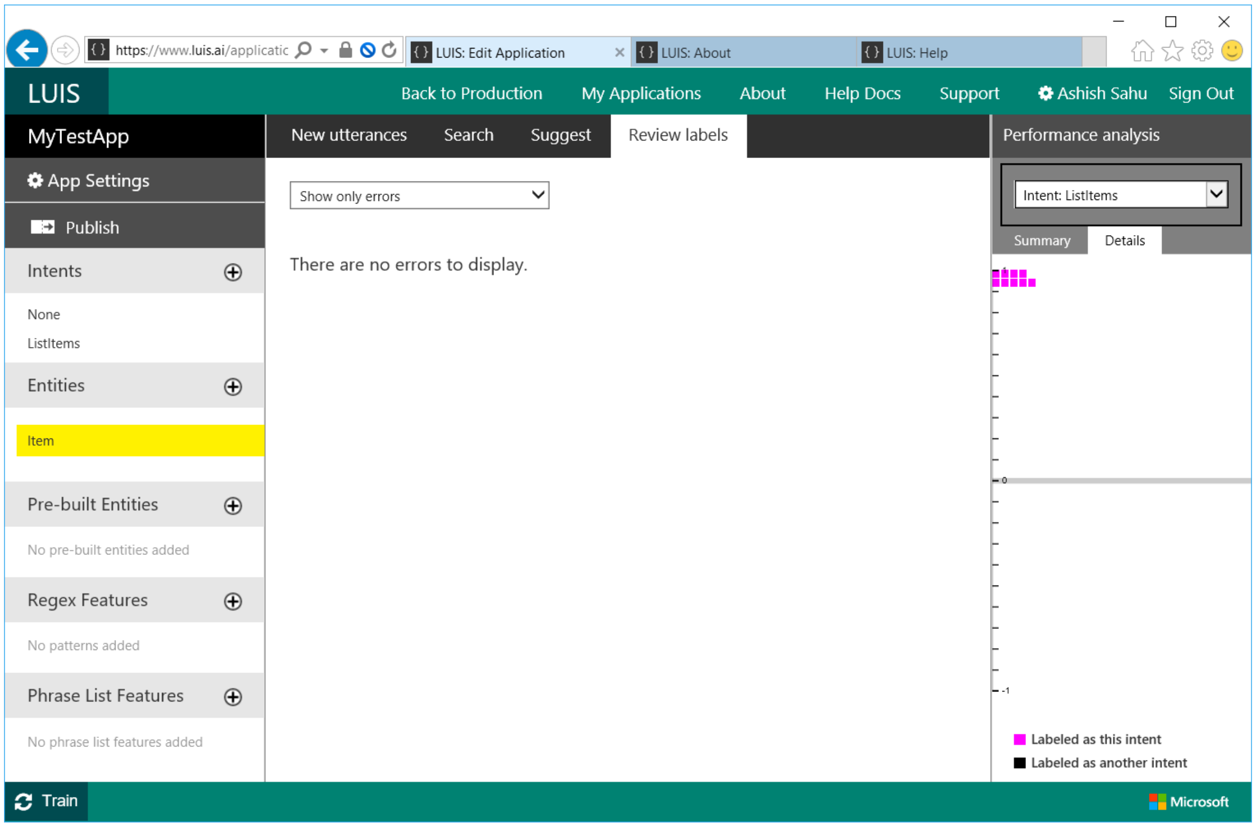
Intents

ListItems
9 utterances: 9 correctly predicted

None
0 utterances: 0 correctly predicted

🟩 Correctly predicted
🟥 Error (predicted as other intent)

Figure 6



There are no errors to display.

Figure 7

You can also get similar information for the entities in your language model.

This information along with the ***Review Labels*** (refer *Figure 6* and *7*) section of the portal can help you look at and analyze any errors with your language model.

Calling LUIS from C# UWP/ASP.NET apps

If you are building a UWP app or ASP.NET web app using C#, you can use the following classes as denoted in the *Figure 8* below to deserialize the JSON response –

```
public class LUISResponse
{
    public string query { get; set; }
    public lIntent[] intents { get; set; }
    public lEntity[] entities { get; set; }
}

public class lIntent
{
    public string intent { get; set; }
    public float score { get; set; }
}

public class lEntity
{
    public string entity { get; set; }
    public string type { get; set; }
    public int startIndex { get; set; }
    public int endIndex { get; set; }
    public float score { get; set; }
}
```

Figure 8

And, the code in *Figure 9* below in your C# UWP or ASP.NET app can use these classes to get the intent and entities information –

```
private async Task LUISParse(string queryString)
{
    using (var client = new HttpClient())
    {
string uri =
       "https://api.projectoxford.ai/luis/v1/application?id=<application-
id>&subscription-key=<subscription-key>&q=" + queryString;
HttpResponseMessage msg = await client.GetAsync(uri);

if (msg.IsSuccessStatusCode)
{
    var jsonResponse = await msg.Content.ReadAsStringAsync();
    var _Data =
       JsonConvert.DeserializeObject<LUISResponse>(jsonResponse);

    var entityFound = _Data.entities[0].entity;
    var topIntent = _Data.intents[0].intent;
}
    }
}
```

Figure 9

Based on your requirements, you can run the response through a loop to extract multiple entities of different types as well the score information about the intents detected in the query string.

Using LUIS with Bot Framework

If you are using our awesome Bot Framework to build a bot and are looking to use LUIS to add natural language intelligence, you will be pleased to know that the ***Microsoft.Bot.Builder*** namespace in the Bot SDK makes it extremely easy to connect with your LUIS application and filter out the intents and entities. In the ***MessageController*** of your bot framework solution, add the following line to route all incoming messages to the class called ***LuisConnect*** –

```
await Microsoft.Bot.Builder.Dialogs.Conversation.SendAsync(activity, () => new
LuisConnect());
```

Now add a class file called ***LuisConnect.cs*** in your project and change the code as shown in *Figure 10* –

```
using System;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using Microsoft.Bot.Connector;

namespace BotApp2
{
    [LuisModel("<application-id>", "<subscription-key>")]
    [Serializable]
    public class Luis: LuisDialog<object>
    {
        [LuisIntent("")]
        public async Task None(IDialogContext context, LuisResult result)
        {
            string message =
                     "I'm sorry I didn't understand. Try asking about stocks or
              inventory.";
            await context.PostAsync(message);
            context.Wait(MessageReceived);
        }

        [LuisIntent("ListItems")]
        public async Task ListInventory(IDialogContext context, LuisResult result)
        {
            string message = "";
            if (result.Entities.Count != 0 && result.Intents.Count != 0 ) message =
$"I detected the intent \"{result.Intents[0].Intent}\" for
\"{result.Entities[0].Entity}\". Was that right?";
            await context.PostAsync(message);
            context.Wait(MessageReceived);
        }


        public async Task StartAsync(IDialogContext context)
        {
            context.Wait(MessageReceived);
        }

    }
}
```

Figure 10

Run your bot locally and try asking questions such as "*Show me the stocks of shirts*" or "*How many belts do we have in stock*" and you should get the appropriate responses with the intents and entities back from the bot.

The most interesting part about the above code snippet is that we just had to label our methods with *[LuisIntent]* and the SDK takes care of calling the LUIS application and getting back results from the LUIS service. This makes it really quick and simple to start adding the language intelligence in our apps.

The Future

The LUIS offering is ready to be used in your apps but it is still being improved and new features are being added frequently. There are some features that aren't covered in this portal but are available for public preview. They are exciting and still in development –

- Integration with Bot Framework and Slack – you can try this out when publishing your LUIS application in Preview Portal. This integration allows for quick LUIS integration with Microsoft Bot Framework and Slack.
- Dialog Support – Dialog support in LUIS allows you to add conversational intelligence in your LUIS application so it can ask for more information from the users on its own if the query requires more information than provided by the users at first. For example, a flight app can prompt for a travel date if the users asks for flight information with just the city name
- Action Fulfillment – this feature allows you to fulfill the user triggered actions using the built-in and custom channel right from your LUIS app.

These features are really exciting and enable more natural conversational interaction in your app with little efforts. They need depth exploration on their own and I hope to do that soon.

Summary

If you are reading this excerpt in this article, I hope you now understand what LUIS can do for you and how effortlessly you can start leveraging it to add a more natural human interaction element to your applications.

In this article, we went through the basics of the LUIS service. We created a LUIS application, built and trained our language model to help us understand what our users mean when they ask something. We also looked at the ways in which this LUIS application can be used from your apps, web services and in your bots. A sample project that contains the LUIS model, UWP app and the Bot sample code mentioned the article can be found on GitHub here –

http://bit.ly/2eEmPsy