Fast and easy language understanding for dialog systems with Microsoft Language Understanding Intelligent Service (LUIS)

Jason D. Williams, Eslam Kamal, Mokhtar Ashour, Hani Amr, Jessica Miller, Geoff Zweig
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
jason.williams@microsoft.com

Abstract

With Language Understanding Intelligent Service (LUIS), developers without machine learning expertise can quickly build and use language understanding models specific to their task. LUIS is entirely cloud-based: developers log into a website, enter a few example utterances and their labels, and then deploy a model to an HTTP endpoint. Utterances sent to the endpoint are logged and can be efficiently labeled using active learning. Visualizations help identify issues, which can be resolved by either adding more labels or by giving hints to the machine learner in the form of features. Altogether, a developer can create and deploy an initial language understanding model in minutes, and easily maintain it as usage of their application grows.

1 Introduction and Background

In a spoken dialog system, language understanding (LU) converts from the words in an utterance into a machine-readable meaning representation, typically indicating the *intent* of the utterance and any *entities* present in the utterance (Wang et al., 2005; Tur and Mori, 2011). For example, consider a physical fitness domain, with a dialog system embedded in a wearable device like a watch. This dialog system could recognize intents like `StartActivity` and `StopActivity`, and could recognize entities like `ActivityType`. In the user utterance "begin a jog", the goal of LU is to identify the utterance intent as `StartActivity`, and identify the entity `ActivityType=''jog''`.

Historically, there have been two options for implementing language understanding: machine-learning (ML) models and handcrafted rules. Handcrafted rules are accessible for general software developers, but they are difficult to scale up, and do not benefit from data. ML-based models are trained on real usage data, generalize well to new situations, and are superior in terms of robustness. However, they require rare and expensive expertise, and are therefore generally employed only by organizations with substantial resources.

Microsoft's Language Understanding Intelligent Service (LUIS) aims to enable software developers to create cloud-based machine-learning language understanding models specific to their application domain, *without ML expertise*. LUIS is built on prior work in Microsoft Research on interactive learning (Simard et al, 2014), and rapid development of language understanding models (Williams et al., 2015).

2 LUIS overview

Developers begin by creating a new LUIS "application", and specifying the intents and entities needed in their domain. They then enter a few utterances they would like their application to handle. For each, they choose the intent label by choosing from a drop-down, and specify any entities in the utterance by highlighting a contiguous subset of words in the utterance. As the developer enters labels, the model is automatically and asynchronously re-built (requring 1-2 seconds), and the current model is used to propose labels when new utterances are entered. These proposed labels serve two purposes: first, they act as a rotating test set and illustrate the performance of the current model on unseen data; second, when the proposed labels are correct, they act as an accelerator.

As labeling progresses, LUIS shows several visualizations which show performance, including overall accuracy and any confusions – for example, if an utterance is labeled with the intent `StartActivity` but is being classified as `StopActivity`, or if an utterance was la-

Proceedings of the SIGDIAL 2015 Conference, pages 159–161,
Prague, Czech Republic, 2-4 September 2015.

beled as containing an instance of the entity `ActivityType`, but that entity is not being detected. These visualizations are shown on all the data labeled so far; i.e., the visualizations show performance on the training set, which is important because developers want to ensure that their model will reproduce the labels they've entered.

When a classification error surfaces in a visualization, developers have a few options for fixing it: they can add more labels; they can change a label (for example, if an utterance was mis-labeled); or they can add a *feature*. A feature is a dictionary of words or phrases which will be used by the machine learning algorithm. Features are particularly useful for helping the models to generalize from very few examples – for example, to help a model generalize to many types of devices, the developer could add a feature called `ActivityWords` that contains 100 words like "run", "walk", "jog", "hike", and so on. This would help the learner generalize from a few examples like "begin a walk" and "start tracking a run", without needing to label utterances with every type of activity.

In addition to creating custom entities, developers can also add "pre-built" ready-to-use entities, including numbers, temperatures, locations, monetary amounts, ages, encyclopaedic concepts, dates, and times.

At any point, the developer can "publish" their models to an HTTP endpoint. This HTTP endpoint takes the utterance text as input, and returns an object in JavaScript Object Notation (JSON) form. An example of the return format is shown in Figure 1. This URL can then be called from within the developer's application. The endpoint is accessible by any internet-connected device, including mobile phones, tablets, wearables, robots, and embedded devices; and is optimized for real-time operation.

As utterances are received on the HTTP endpoint, they are logged, and are available for labeling in LUIS. However, successful applications will receive substantial usage, so labeling every utterance would be inefficient. LUIS provides two ways of managing large scale traffic efficiently. First, a conventional (text) search index is created which allows a developer to search for utterances that contain a word or phrase, like "switch on" or "air conditioning". This lets a developer explore the data to look for new intents or entirely new


```
{
  "query": "start tracking a run",
  "entities": [
    {
      "entity": "run",
      "type": "ActivityType"
    }
  ],
  "intents": [
    {
      "intent": "StartActivity",
      "score": 0.993625045
    },
    {
      "intent": "None",
      "score": 0.03260582
    },
    {
      "intent": "StopActivity",
      "score": 0.0249939673
    },
    {
      "intent": "SetHRTarget",
      "score": 0.003474009
    }
  ]
}
```


Figure 1: Example JSON response for the utterance "start tracking a run".

phrasings. Second, LUIS can *suggest* the most useful utterances to label by using active learning. Here, all logged utterances are scored with the current model, and utterances closest to the decision boundary are presented first. This ensures that the developer's labeling effort has maximal impact.

3 Demonstration

This demonstration will largely follow the presentation of LUIS at the Microsoft //build developer event. A video of this presentation is available at `www.luis.ai/home/video`.

The demonstration begins by logging into `www.luis.ai` and inputting the intents and entities in the domain, including new domain-specific entities and pre-built entities. The developer then starts entering utterances in the domain and labeling them. After a label is entered, the model is re-built, and the visualizations are updated. When errors are observed, a feature is added to address them. The demonstration continues by publishing the model to an HTTP endpoint, and a few requests are made to the endpoint by using a second web browser window, or by running a Python script to simulate more usage. The demonstration then shows how these utterances are now available for labeling in LUIS, either through searching, or

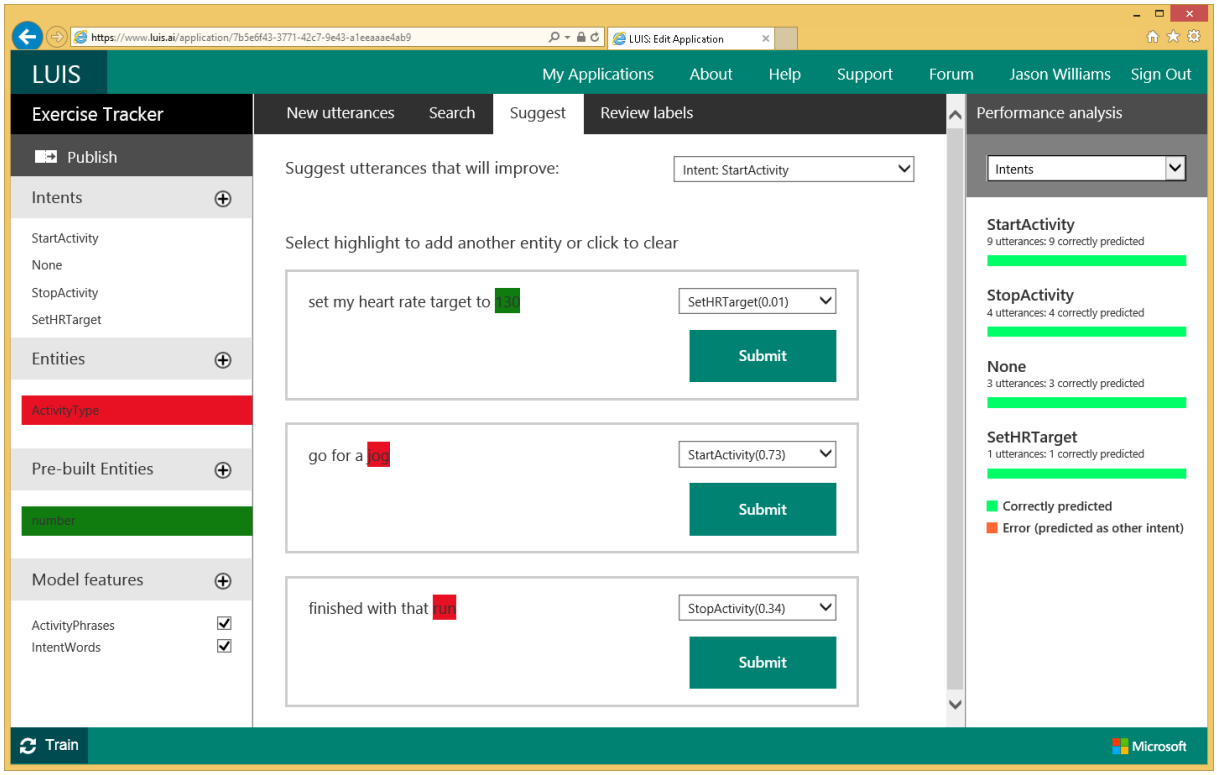

Figure 2: Microsoft Language Understanding Intelligent Service (LUIS). In the left pane, the developer can add or remove intents, entities, and features. By clicking on a feature, the developer can edit the words and phrases in that feature. The center pane provides different ways of labeling utterances: in the "New utterances" tab, the developer can type in new utterances; in the "Search" tab, the developer can run text searches for unlabeled utterances received on the HTTP endpoint; in the "Suggest" tab, LUIS scans utterances received on the HTTP endpoint and automatically suggests utterances to label using active learning; and in the "Review labels" tab, the developer can see utterances they've already labeled. The right pane, shows application performance – the drop-down box lets the developer drill down to see performance of individual intents or entities.

by using active learning. After labeling a few utterances using these methods, the demonstration concludes by showing how the updated application can be instantly re-published.

4 Access

LUIS is currently in use by hundreds of developers in an invitation-only beta – an invitation may be requested at `www.luis.ai`. We have begun in an invitation-only mode so that we can work closely with a group of developers of a manageable size, to understand their needs and refine the user interface. We expect to migrate to an open public beta in the coming months.

References

P Simard et al. 2014. ICE: Enabling non-experts to build models interactively for large-scale lopsided problems. `http://arxiv.org/ftp/arxiv/papers/1409/1409.4814.pdf`.

G Tur and R De Mori. 2011. *Spoken Language Understanding — Systems for Extracting Semantic Information from Speech*. John Wiley and Sons.

Y Wang, L Deng, and A Acero. 2005. Spoken language understanding. *Signal Processing Magazine, IEEE*, 22(5):16–31, Sept.

JD Williams, NB Niraula, P Dasigi, A Lakshmiratan, CGJ Suarez, M Reddy, and G Zweig. 2015. Rapidly scaling dialog systems with interactive learning. In *International Workshop on Spoken Dialog Systems, Busan, Korea*.