Intents in Natural Language Processing

Intent Creation using the IIUSA Intent Editor

Miles Murdocca (IIUSA)
murdocca@iiusatech.com
http://milesmurdocca.com

Object Pascal editor by Juan Jiménez
Python editor by Miles Murdocca

Synopsis

An *intent* is an action that a chatbot user wants to perform: book a flight, order a pizza, get assistance with a task. Intents are used pervasively in user input processing for chatbots and in natural language processing (NLP) tasks.

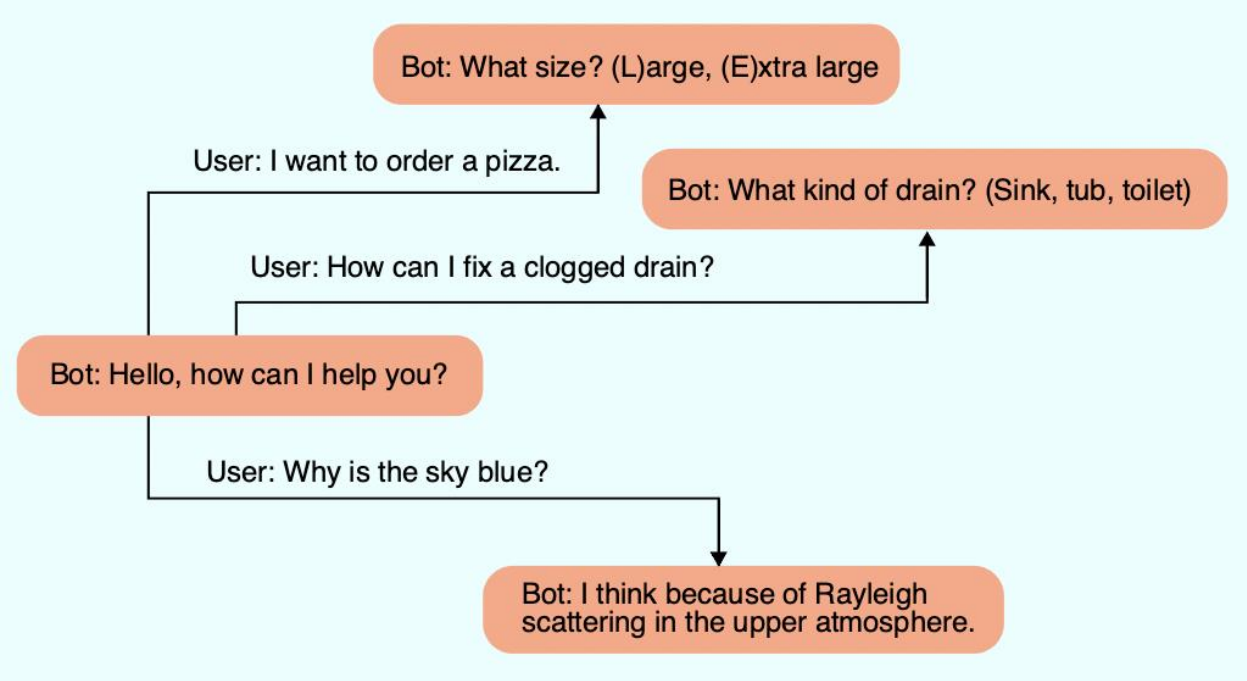

Figure 1: Chatbot conversation flow.

There are several chatbot services that help the user create a chatbot through a graphical user interface (GUI). For those of us who create our own chatbots, we need a tool to create and edit our own pool of intents. A simple spreadsheet can work in lieu of a software tool, but is tedious and prone to error.

This paper covers the freely available IIUSA Intent Editor and how to integrate the JavaScript Object Notation (JSON) intent files into a conversational chatbot.

Topics covered:

1) Rule-based static intents
2) Dynamic intents
3) Matching user input to intents

4) Representing intents
5) IIUSA Intent Editor
6) Putting it all together: working example
7) Conclusion

1. Rule-based static intents
In a simple scenario, a chatbot matches user intents and responds according to a set of rules. The responses can be simple one-turn text relationships:

```
User: What is the capital of Indonesia?
Bot: Jakarta
```

These simple one-turn (no memory of previous inquiries) intents can be extended with simple statically resolved substitutions, in which the substitution choices are stored in a table:

```
User: What is the capital of {{PlaceName}}
Bot: {{PlaceLookup(%%PlaceName%%)}}
```

Country	Capital	Continent	Languages
India	New Delhi	Asia	Hindi and English
Indonesia	Jakarta	Asia	Indonesian
USA	Washington, D.C.	North America	American English
China	Beijing	Asia	Mandarin
Russia	Moscow	Europe and Asia	Russian
England	London	Europe	English
Peru	Lima	South America	Spanish and Aymara

Figure 2: Lookup table for `PlaceName` substitutions.

The User input and the substituted Bot output shown farther above are derived from the lookup table.

2. Dynamic intents
Building on our lookup table capability, a next-level bot might implement dynamic one-turn responses that change based on changing situations. For example, we can represent the day of the week in a coded format such as:

{lambda x: day_of_week()}

and then have our chatbot automatically execute the so-called *lambda function* when a *trigger* element matches, to make the substitution:

```
User: What day is it?
Bot: Wednesday
```

For this, we use a different kind of table for lambda functions, shown in Figure 3:

Trigger Name	Substitution
which-day	{lambda x: day_of_week()}
do-a-search	{lambda x: google(x)}
which-is-larger	{lambda x,y: max(x,y)}
are-you-happy	{lambda x: “yes”}
[...]	

Figure 3: Lookup table including a mix of lambda functions.

Now given our pool of static and dynamic intents, how can our chatbot decide which intent is the best match?

3. Matching user input to intents

We want to send the user the best reply, and our approach is to make the best match between what the user types or utters and our intent. So how does our bot decide?

One possibility is to calculate similarity based on the *Levenshtein distance*, also known as the *edit distance.* The Levenshtein distance is a measure of the fewest character changes needed to convert the subject word into a word in the vocabulary, and is often used in spelling correction. It is named after the Soviet mathematician Vladimir Levenshtein, who considered this distance in 1965.

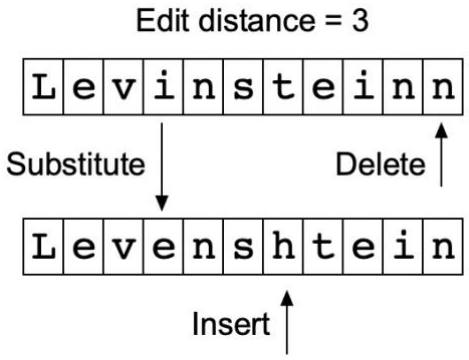

Figure 4: An example of Levenshtein distance.

This can work well for word correction, but not so well for phrase matching which might be missing an entire word or have an extraneous word.

For our purpose of phrase matching, we use *cosine similarity* to determine how well the user’s inquiry matches an entry in our intent pool. The goal is not to find the best response to the user’s input, but to find the best match between the user’s inquiry and an intent our bot knows about, and then return the response for that match.

Cosine Similarity
We can convert our phrases into numerical vectors with a technique known as *term frequency – inverse document frequency* (TF–IDF). This is a transformation applied to text strings to get two vectors that represent the two strings.

Term Frequency: is a scoring of the frequency of the word in the current document.
TF = (Number of times term *t* appears in a document)/(Number of terms in the document)

Inverse Document Frequency: is a scoring of how rare the word is across documents.
IDF = 1+log(*N*/*n*), where *N* is the number of documents and *n* is the number of documents a term *t* has appeared in.

The TF-IDF weight is more generally used in information retrieval and text mining. It is a statistical measure that evaluates how important a word is to a document in a collection (also known as a *corpus*).

We can then obtain the Cosine similarity of any pair of vectors *v1* and *v2* by taking their dot product and then dividing that by the product of their norms (which normalizes the vectors in a mathematical sense):

$$\text{CosineSimilarity}(v1, v2) = \text{DotProduct}(v1, v2) / ||v1|| * ||v2||$$

As a regular formula, in which $\boldsymbol{A} = v1$ and $\boldsymbol{B} = v2$:

$$similarity = cos(\theta) = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{||\boldsymbol{A}|| \, ||\boldsymbol{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

This yields the cosine of the angle θ between the vectors. This allows us to find the similarity between two phrases represented as vectors *v1* and *v2*.

Figure 4 shows a simplified three dimensional space for illustration. In practice, the vector space has as many dimensions as there are unique words in all sentences combined. So if our vocabulary has 1000 words, then there will be 1000 dimensions in the vector space.

In the case of information retrieval, the cosine similarity of two phrases will range from 0 to 1, since the term frequencies cannot be negative. Likewise the angle between two term frequency vectors cannot be greater than 90°.

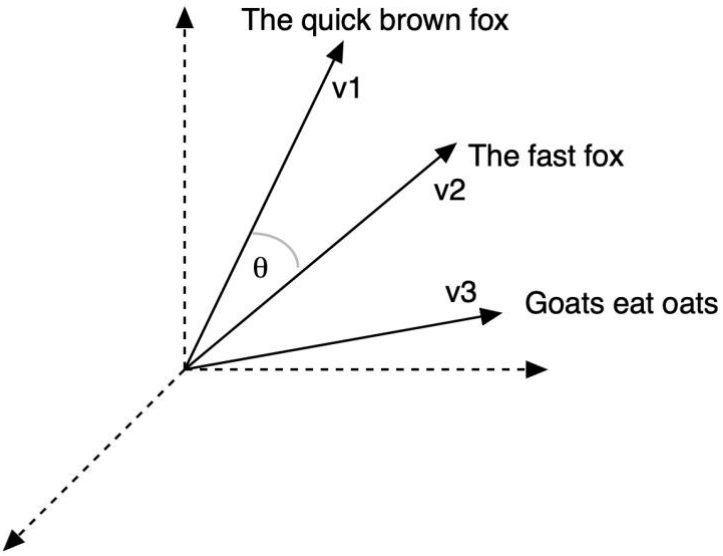

Figure 4: The vector space model and cosine similarity.

A good discussion on cosine similarity:
https://towardsdatascience.com/calculating-string-similarity-in-python-276e18a7d33a

4. Representing intents
We created a pool of intents:

https://iiusatech.com/downloads/intent-editor/intents.json

The intents are in JSON format depicted below, and you can ignore the syntax if you use the IIUSA Intent Editor:

```
{
  "intents": [
    {
      "tag": "name",
      "patterns": [
        "what's your name?",
        "what is your name",
        "what are you called?",
        "who are you?"
      ],
      "responses": [
        "My name is Reddy",
        "I'm Reddy",
        "I am Reddy",
        "My name is Reddy"
      ]
    },
    {
      "tag": "ruhuman",
      "patterns": [
        "Are you real?",
        "Are you human",
```

```
        "Are you a robot?",
        "Are you a machine?"
      ],
      "responses": [
        "I am as real as Data gets.",
        "I am kinda \"choose your own adventure.\"",
        "I aim to be."
      ]
    },
    {
      "tag": "time",
      "patterns": [
        "What time is it?",
        "Do you know the time?",
        "Do you have the time?"
      ],
      "responses": [
        "Check your monitor.",
        "It's the same time as it was yesterday at this time.",
        "All time is relative."
      ]
    }
  ]
}
```

The intents have three parts: a Tag (name of the intent), Patterns (things a user might type or utter), and Responses (what the bot replies back). The pool is created for conversation banter.

We use a 70% cosine similarity score to decide whether to use the intent response from our pool of intents, or to send the user input to the next higher tier, which could be a human in a business use case, but for our case is a GPT-2 conversational bot with customized parameters:

https://iiusatechai.com/demos/demobot/mapper

A question like “Who are you?” is a 100% match for the `name` intent in our pool, and one of the five intent responses are randomly selected, for this case “My name is Reddy”:

```
User: Who are you?
```

*[**result_table: (1.0000) My name is Reddy] [**result_GPT: Bot: I am the one who knocks. Reference question for table lookup: who are you]*

[*n.g.* The GPT-2 response comes from a line in the *Breaking Bad* series.]

A variation that is less than a 100% match but still above our 70% threshold is “Your name is?”:

```
User: Your name is?
```

*[**result_table: (0.8660) I'm Reddy] [**result_GPT: Bot: I am the one who knocks. Reference question for table lookup: what is your name]*

A variation that results in only a 61% match is “How do you like to be called?”:

```
User: How do you like to be called?
```

*[**result_GPT-2: Bot: I am the one who knocks.] [**result_table: (0.6172) In your head, maybe. Reference question for table lookup: where do you like to go]*

The results have been reordered by the bot to put the better response first.

5. The open-source IIUSA intent editor

You can download our freely available intent editor to create and edit intent files in JSON format:

https://iiusatech.com/intent/

Figure 5 shows a screenshot of the editor, Object Pascal edition. See farther down for the Python edition.

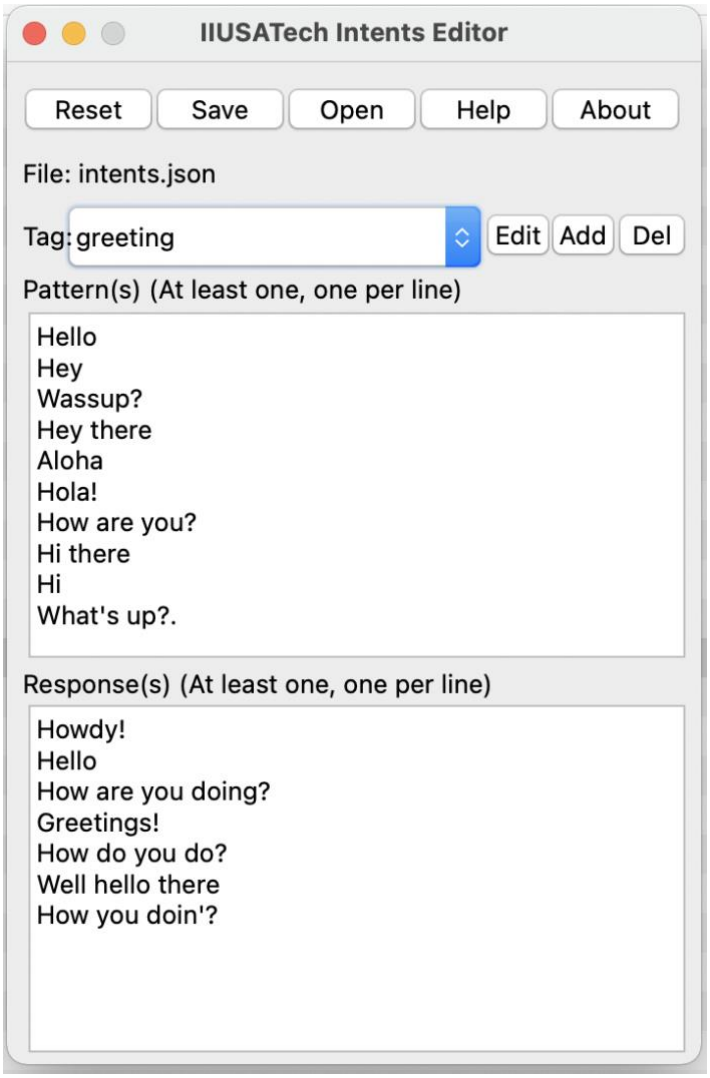

Figure 5: The IIUSA Intent Editor – Object Pascal edition.

The Object Pascal editor is an application (app) for Windows 32-bit and 64-bit operating systems, as well as Apple OSX 64-bit (Intel processors only at this point). Implementations are slated in the future for Android and iOS apps. The code is written in Object Pascal and is meant to be built with Embarcadero's RAD Studio Delphi, of which there is a free community edition for non-commercial use (see below for download link).

To load an existing intent pool, such as the intents.json file introduced in the previous section, click the Open button, choose the file and load it. You can then add, edit, and delete tags, as well as create and edit groups of patterns and responses.

To create a new intent file, start with the default empty pool or click the Reset button, and Add new tags, patterns, and responses. You can save the intents in a JSON file and reload it at any time to continue working on it.

The Help button opens your default browser and takes you to a help file on the IIUSA Website, and the About button tells you what version of the app you are using. That's all there is to it!

To download the apps or if you would like to tinker with the Object Pascal source code or maybe contribute to the project, it is open source and available on GitHub:

https://github.com/IIUSA/intents-editor-app

Keep in mind *we do not provide support on learning how to use RAD Studio Delphi or the details of the Object Pascal language*. In addition, to produce an OSX standalone executable you will have to provide your own Apple Developer Certificate. You can download the community version of the development system we use here:

https://www.embarcadero.com/products/delphi/starter/free-download

If you prefer a Python intent editor, have a look here for an entirely different codebase:

https://iiusatech.com/downloads/intent-editor/simplepython/

The screenshot in Figure 6 shows the editor layout.

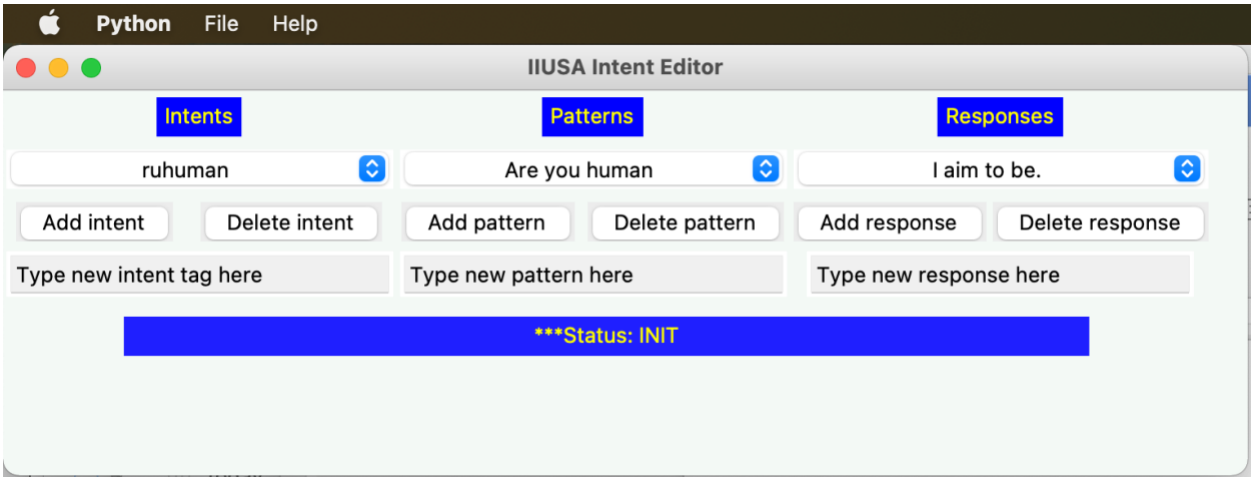

Figure 6: The IIUSA Intent Editor – Python edition.

By default, the editor loads Data/intents.json. You can load another file and also save your changes through the File menu.

After the program opens, start by selecting an intent from the Intents dropdown. Then move to the Patterns and Responses dropdowns to view the existing patterns and responses.

Use the three textboxes and the three Add buttons to add intents, patterns, and responses. Use the three Delete buttons to delete intents, patterns, and responses.

Currently there is no Edit function: if you want to change a pattern or response, you will need to add the new one and delete the old. The intent names are not editable.

6. Putting it all together: a working example

You can try the elements in this article with the Conversational AI chatbot on the IIUSA demo page:

https://iiusatech.com/demos/

You will need to use a few tricks described below to expose hidden elements.

The chatbot as shown on the page is a pass-through to a GPT-2 model with default parameters on a huggingface.co download of the microsoft/DialoGPT-large model. You can invoke the intent mapper and filter 70% or greater cosine similarity matches by preceding an utterance with "`qaz `" (this is not case sensitive; note the space after qaz).

The score shows the cosine similarity, which is 100% for the example below. Try a few variations to see how the similarity score changes with the qaz hidden keyword like:

```
You: What is your name?
Bot: I'm not sure what you mean

You: qaz What is your name?
Bot: My name is Reddy [100%]

You: qaz Your name is?
Bot: My name is Reddy [87%]
```

The `qaz` hidden keyword returns the best match 70% or greater for the intent mapper. Lesser matches are passed back to GPT-2. The hidden keyword "`qae `" (case insensitive) forces the intent mapper response to be taken, even if worse than 70%:

```
You: How do you like to be called?
Bot: I like to be called a lot.

You: qwe How do you like to be called?
Bot: In a galaxy far, far away. [67%]
```

7. Conclusion

Intents capture a basic pattern-response characteristic of conversational chatbots. Chatbot platforms have built-in intent editors, but if you are creating intents outside of the platform, you will need a way to create and store intents.

Use the IIUSA Intent Editor to create intents for your own chatbot. The code is open source and you can use and modify it as you wish.