Stella: Problem Solving with Conversational Agents

Dante Tam, 5/29/2017

In this article, we work through a complicated, intricate computer science problem. We begin by analyzing it, brainstorming and implementing solutions, and then testing those solutions on clearly defined tests and metrics, while maintaining best practices and good software architecture.

Stating the problem: My conversational agent, Stella, takes in a text command, and needs to return the appropriate action. Given a list of actions, this is a relatively easy task for a human. However, it is much harder for computers to see grammatical structures, word contexts, etc.

In machine learning, this is called a classification problem. For review, a classification asks given some data point X_i, what is the actual class or category y_i? In this case, our input data is a sentence, and our possible class choices are the various commands. For example,

$$\text{"} Please\ show\ my\ finances. \text{"} \Rightarrow\ the\ finance\ command$$

First Approach, Text Similarity: We model every command as having some important keywords. The more keywords the command and input share, the more likely the command is correct. We extend this to include synonyms of the words present. Technically, we achieved this using WordNet, a collection of English words organized into a graph. Words that are similar to each other i.e. synonyms, are connected together. We analyzed the similarity of two words by their degrees of connection. Much like how LinkedIn shows you more of your closer friends and 1st degree connections, we prioritized first degree synonyms. We also give less but significant priority to similarity of defintions and sentence usages i.e. dictionary entries.

Second Approach, Zero-One Encoded Vectors w/ Softmax Regression: We represent the inputs as vectors, and the commands as numbered labels. The problem transforms below, from the first type to the second:

$$\text{"} Please\ show\ my\ finances. \text{"} \Rightarrow\ the\ finance\ command$$
$$\ldots$$
$$[0, 0, 1, 0, 1, 1, 1, \ldots]\ \Rightarrow 17$$

Simply put, we want to find a device that turns the sentence into a label. Mathematically, our goal is to find a vector w such that

$$Xw = \hat{y} \approx y \Leftrightarrow minarg_w ||Xw - y||_2^2$$

where $\hat{y}$ is our best guess. We use a method called softmax regression which achieves close to the goal. Google's machine learning library TensorFlow helps us greatly with this task.

To get the sentences as vectors, we collect all the unique words, numbered 1 to V. Then every sentence can be represented as a vector s of length V, such that $s_i = 1$ means that the sentence contains the ith word, otherwise $s_i = 0$. For example,

$$"Please\ show\ my\ finances.",\ "Show\ my\ citations."$$
$$Please \Leftrightarrow 1, show \Leftrightarrow 2, my \Leftrightarrow 3, finances \Leftrightarrow 4, citations \Leftrightarrow 5$$
$$[1, 1, 1, 1, 0],\ [0, 1, 1, 0, 1]$$

Overall, this approach had near 73% validation accuracy, and while quite neat, was not strong enough for our applications.

Third Approach, Convolutional Neural Networks for Sentence Classification: We use a convolutional neural network (CNN) on a large sentence vector, as described in the research paper of the same name (Yoon Kim, 2014). We use word2vec, a tool developed which converts words into vectors of length k, such that similar words have similar vectors. Sentences are now n by k matrices, where n is the maximum length of all sentences and k is the dimension from earlier. This essentially gives us an image, which is processed well by CNNs. For regular images, CNNs can become edge detectors, brightness detectors, and so on, just as our CNN can begin to recognize large, semi-abstract concepts such as grammatical structures, word contexts, etc.

This method was state of the art in 2014 and not surprisingly it fared well, scoring 97% training accuracy.

We finally have an algorithm for our conversational agent, Stella, which takes in user requests and completes the appropriate action. It is a robust solution, scoring very highly on our validation tests. It is extensible, as we can add many more classes and commands and training data. It is organized into its own Pythom module, which can be called in a desktop application, web server, etc. It is a good solution.