Better Glue for Pipelines

504: Machine Learning meets Program Analysis Assignment 1

Luheng He

luheng@cs.washington.edu

Softwares used to solve natural language processing (NLP) and machine learning (ML) tasks are usually pipelined and consist of a series of subtasks. In Table 1, the subtasks in blue color are mostly task-independent, and usually harder to develop. For these tasks, people usually depend on well-developed open-source softwares, such as NLTK, Stanford NLP (for tokenization, tagging, parsing, etc.) and LibSVM, Mallet, Torch (for learning). The other subtasks, such as input reader, feature extraction and evaluation, are highly task-dependent and have to be developed from scratch. Sometimes people are required to use a published evaluation script (in Perl, for many NLP tasks) for the purpose of fair comparison. For NLP tasks in particular, due to their structured nature and huge input/output space, there is really no ideal software that fits the requirement of all tasks.

	Typical subtasks for NLP	**Typical subtasks for ML**
1	Input Reader	Input Reader
2	Segmentation/tokenization	Pre-processing/Data filtering
3	Pos-tagging/Parsing/Named-entity Recognition	
4	Feature Extraction for the target task	
5	Parameter Fitting (Learning)	
6	Evaluation/Cross validation	
7	Hyper-parameter Tuning/Model Ensemble	
8	Output/Analysis/Visualization	

Table 1: Sub-tasks in NLP/ML pipelines.

Therefore, in a typical pipelined NLP/ML software, developers are usually forced to alternate between 1). software written by themselves and by someone else. 2). software written in

different programming languages. Glue code are written to pass around data and parameters. For example, someone would write a some data processing script in Python to scrape all the English titles from Wikipedia, dump them into a file, use Stanford NLP (in Java) to parse the titles and store the results into a list of objects, write some Java code to extract features, learn a classifier using the Java LibSVM wrapper. And for evaluation, what one might do is write the prediction in a text file and run a Perl script from the commandline.

There are many things that can be improved in this pipelined process. The most important one is probably about the glue code written to connect the subtasks. Glue code is usually developed in a hurry, lack developers' attention, and almost never get tested or verified.

Here is a realistic case: When using files to pass around intermediate data, developers need to keep track of the file format. Sometimes Mthe file format is specified in the documentation, sometimes in comments such as /* {word} \t {tag} \t f1, f2 …. */, and sometimes in the developers' brains. Misunderstanding and misalignment in file formats cause errors. For example, the syntax parsing software outputs a parse tree with word indices starting from 1 but I thought they start from 0.

Possible improvements to this problem would include:

1. Write comprehensive comments/documentation for the Input/Output file formats or objects.
2. Write unit tests for the Reader/Writer functions in the software.
3. Write methods to check the validity of input files/objects.
4. Provide sample Input/Output data along with the software.

However, doing one or more of the above suggested could slow down development. It would be great to come up with tools to automatically generate/suggest test cases/sanity checks for the Reader/Writer methods. Maybe we can come up with an easy way for developers to specify the types and the constraints of the intermediate data objects between the subtasks, and automatically generate tests/verifications to check those types and constraints.