Document Analysis Pipeline

For BioNLP, Spring 2012

Michael Shafir, Michael.Shafir@gmail.com, Brandeis University

Architecture

The core foundation for this pipeline is the idea of layers of representation extended by stages of processing

Layer of Representation

A representation is made of numerous layers. For example, one can have a token layer, a vocabulary layer, a document layer, etc. A representation layer is basically some index into the data.

Each layer is made of multiple indexes into the data and each of these can have properties or features. For instance, a vocab layer is made of multiple indexes with the keys being unique words and values being the features or properties of those words. Total count in the data can be one such feature.

It can be confusing, but the representation is essentially a **3 layer hash table**. If we want to get the count for the word "the" in the text, it would look like: representation["VocabLayer"]["the"]["count"]. The "get" and "set" convenience functions can make this process simpler:

- representation["VocabLayer"].get("the","count")
- representation["VocabLayer"].set("the","count",5)

representation["VocabLayer"].add("the") adds a new key for the word "the" to the vocab layer, the value will be an empty dictionary.

Representations save out to and load from xml files or the db depending on how the parameter file is set up.

To add a new representation layer:

1. Create a new python file in the "layers" folder.
2. Use the following skeleton (be sure to import * from representation)

```
class LayerName(RepresentationLayer):
    def create(self,db,rep):
        #use self.add(key) to add the keys for that layer
        #alternatively, you can use self.set(key,feature,value)
        #this will both add the key and set a value for a feature
        #of that key
```

Crucially, your class name must be the same as the file's name (minus the .py extension)

Stages of Processing

A stage of processing defines some operation over the representation layers. Usually (but not always) this involves adding a new feature to a representation layer. Stages can also be used for outputting data in a specific format or running shell commands. Later we can also envision visualization stages.

To create a stage, use the following template:

```
from stage import *

class StageName(Stage):
    def __init__(self,name):
        Stage.__init__(self,name,'LayerToProcess','FeatureToAdd')

    def pre_process(self,db,rep):
        pass

    def process(self,key,features,db,rep):
        return [value for added feature]

    def pre_process(self,db,rep):
        pass
```

The pre_process and post_process methods are optional to override, they are executed once before and after (respectively) main processing. The main **process** method is executed once for every key in the specified 'LayerToProcess' so the layer must already have been created before executing a stage over it. The process method provides the key it is currently expecting a new feature value for, the current existing features for that key, and the database and entire representation so you can access all the information known to the system.

The best way to become comfortable with stages and representations is to look at examples of what already exists.

Stages provides a function **shell_command**, which executes a shell command and pipes input and output appropriately. Later, this may be used to also log input and output, so it is advised to use this function.

Pipeline

Running the pipeline requires the following pieces. A pipeline is started via command line with "python processing_pipeline.py <parameter file>"

Stage Ordering File

A stage ordering file specifies what the pipeline will do during any given run. Each line specifies a command (excluding empty lines or # commented lines), to be executed. The possible commands are:

- create [Layer Name] – makes layer Layer Name
- stage [Stage Name] – runs stage Stage Name
- save – saves the full current representation to db or file depending on parameter settings
- load – loads the full representation from db or file depending on parameter settings
- clear – clears the representation from db. Saving rep to db will create duplicate rows each time you do it unless you clear in between.

Because the pipeline supports saving and loading full representations, you can stop and restart the the pipeline at any point to avoid repeating any processing. **The pipeline saves the full representation at the end by default.**

Parameter Files

The pipeline can require many different parameters that control how the stages will run. The syntax of a parameter file is just multiple lines of [parameter] = [parameter value]. Empty lines and # comments are allowed. When the pipeline starts, all parameter settings are loaded into a representation layer called 'Global'

Two convenience functions simplify the process of accessing parameter settings.

- **get_required_param(parameter_name, representation)** – this gets a parameter setting. If that setting has not been specified, the program shows an error message and quits.
- **get_optional_param(parameter_name, representation, default_value=None)** – this gets a parameter setting if it has been specified. Otherwise, it returns 'default_value'