Customizing Quails

This page contains two brief tutorials:

1. Understanding `quails.py`.
2. How to add NLP services to the Quails server.

Understanding `quails.py`

Quails Constants

Quails constants contain hardcoded paths and other static items that are depended on by the interface and the server.

```
# constants
CONFIG_FILE = "config/quails.config"
SERVER = "http://localhost:5000/quails"
DELIMITER = "+"
TRAINING_FILE = "training_questions/train_1000.label"
```

NLP Data Structures

In the configuration file `config.py`, the order of the NLP steps determines their order in the NLP pipeline (see Ask a Question). The structure `valid_pipelines` represents the list of valid NLP pipelines. The server checks against this list when an NLP request comes in. If the incoming pipeline is valid, the request is processed. Otherwise, the server returns a failure message. If you add a new function to `nltkfuns.py` or `stanfuns.py`, you will need to create new pipeline objects that include your new service. This will be demonstrated in the second tutorial on this page. Be careful to consider what type of input your function needs. For example, in the second tutorial we will add a function to the server that extracts noun phrases from parse trees. If we have not already generated a parse tree, then the noun phrases function will fail.

```
valid_pipelines = [['tokens'],['tokens','pos'],
                   ['tokens','pos','ner'],
                   ['tokens','pos','ner','headchunks'],
                   ['tokens','pos','headchunks','ner'],
                   ['tokens','pos','ner','headchunks','parsetree'],
                   ['tokens','pos','headchunks','ner','parsetree'],
                   ['tokens','pos','headchunks','ner','parsetree','deps'],
                   ['tokens','pos','ner','headchunks','parsetree','nounphrases'],
                   ['tokens','pos','headchunks','ner','parsetree','nounphrases']
                  ]
```

The `step_input` structure specifies the input type required by each of the available steps. This is where we specify that the function `nounphrases` requires input type `parsetree`. Naming the input type after the function that generates it is an easy way to manage the dependencies. Moreover, it is required in order for the server to feed the correct input to the current NLP step. This is because when a step is run, say `pos`, the results are stored in a dictionary entry called `pos`. Therefore, when we wish to use the named entity recognizer, `ner`, we can feed it the result of the `pos` function. This can be expressed as the following:

```
# the server calls the function in a different manner...
# this is some funky pseudocode to get the point across

question.nlp_features['ner'] = ner(question.nlp_features['pos'])
```

```
step_input = {'tokens':'text',
              'pos':'tokens',
              'ner':'pos',
              'headchunks':'text',
              'parsetree' : 'text',
              'dep' : 'text',
              'nounphrases' : 'parsetree'}
```

Classification Data Structure

Quails attempts to predict the answer type of the input question. For example, if the question is “How fast is the fastest car?”, Quails determines that the answer required is a number, more specifically a speed. See this page for more information on the Li taxonomy of classification.

The Li classes are encoded in the `classes` dictionary, where the coarse grained class is the key, and the values are each of the fine grained classes that fall under the key.

```
classes = {'ABBR':['abb','exp'],
           'ENTY':['animal','body','color','cremat','currency','dismed',
                   'event','food','instru','lang','letter','other',
                   'plant','product','religion','sport','substance',
                   'symbol','techmeth','termeq','veh','word'],
           'DESC':['def','desc','manner','reason'],
           'HUM':['gr','ind','title','desc'],
           'LOC':['city','country','mount','other','state'],
           'NUM':['code','count','date','dist','money','ord','other',
                  'period','perc','speed','temp','volsize','weight']}
```

Quails Objects

Quails objects are special objects used primarily by the pipeline interface. `run.py`. To view the definitions of the objects, please look in `quails.py`.

Question

The Question class is a representation of the question. The class is defined in `quails.py` and contains a number of question-specific data structures and methods. The data structures hold information about the question, gathered during the question analysis phase, as well as candidate answers and their scores.

QuailsConfig

The QuailsConfig class configures the question answering pipeline by reading the user’s prefences from `config/configquails`. It inherits methods from the python module `configparser`.

How to add NLP Services to the Quails Server

In this tutorial, we will walk through the process of adding an NLTK function called `nounphrases` to the server. The `nounphrases` function will extract a list of noun phrases from the questions parse tree.

This tutorial assumes that you have read the first tutorial and will not re-explain the reasons for adding reference to the `valid_pipelines` and `step_input` structures.

1. Add the function to `nltkfuns.py`.

First, add the function to the list of functions at the top of the file.

```
def nounphrases(input):
        # accepts a dependency tree string
        # returns a list of nounphrases
        # noun phrases are represented as lists of words

        tree = nltk.Tree.fromstring(input)
        subtrees = tree.subtrees(filter = np_filter)
        nounphrases = []

        for st in subtrees:
                pos = st.pos()
                np = ""
                for p in pos:
                        if p[1] != 'DT':
                                np += p[0] + " "

                nounphrases.append(np)

        return nounphrases
```

Second, add the name of the function to the `nltkfuns` list. This list allows the server to build a list of callable functions, or NLP pipeline, based on the user's preferences.

```
nltkfuns = {'tokens' : tokens,
            'pos' : pos,
            'ner' : ner,
            'headchunks' : headchunks,
            'nounphrases' : nounphrases}
```

2. Add the function name to the list of valid pipelines in `quails.py`.

```
valid_pipelines = [['tokens'],['tokens','pos'],
                   ['tokens','pos','ner'],
                   ['tokens','pos','ner','headchunks'],
                   ['tokens','pos','headchunks','ner'],
                   ['tokens','pos','ner','headchunks','parsetree'],
                   ['tokens','pos','headchunks','ner','parsetree'],
                   ['tokens','pos','headchunks','ner','parsetree','deps'],
                   ['tokens','pos','ner','headchunks','parsetree','nounphrases'],
                   ['tokens','pos','headchunks','ner','parsetree','nounphrases']
                  ]
```

3. Add the input type to `quails.py`.

```
step_input = {'tokens':'text',
              'pos':'tokens',
              'ner':'pos',
              'headchunks':'text',
              'parsetree' : 'text',
              'dep' : 'text',
              'nounphrases' : 'parsetree'}
```

Currently, there is no way to streamline this process, but using this guide should make it fairly simple to introduce new NLP functionality to the system.

Future versions will enable the user to add functionality to more components of the Quails system.