Creative Web Services with Pattern

Tom De Smedt
Experimental Media
Research Group (EMRG)
St Lucas Univ. College, Antwerp
tom.desmedt@kdg.be

Lucas Nijs
Experimental Media
Research Group (EMRG)
St Lucas Univ. College, Antwerp
lucas.nijs@kdg.be

Walter Daelemans
Computational Linguistics
Research Group (CLiPS)
University of Antwerp, BE
walter.daelemans@uantwerpen.be

Abstract

Pattern is a Python toolkit for web mining, natural language processing, machine learning, network analysis and data visualisation. In this paper, we discuss how it can be useful as a computational creativity tool, in particular how its new pattern.server module can be used to set up creative web services.

Introduction

Pattern (http://www.clips.ua.ac.be/pattern) is a Python 2.5+ toolkit for web mining, natural language processing, machine learning, network analysis and data visualisation. It is organised in different modules that can be intermixed. For example, the pattern.web module can be used to retrieve Google results, Wikipedia and Wiktionary articles, DBPedia triples, Twitter and Facebook statuses, to crawl and parse HTML, and so on. The pattern.en module has an English part-of-speech tagger, sentiment analysis, regular expressions for inflecting nouns and verbs, and so on. The pattern.vector module contains machine learning tools for classification (e.g, k-NN, SVM), clustering (e.g., *k*-means), dimensionality reduction, feature selection, and so on. The pattern.graph module has tools for network analysis, and for network visualisation using Pattern's canvas.js helper module for interactive graphics in the web browser. For an overview, see De Smedt & Daelemans (2012).

In recent years, the functionality has steadily expanded. Pattern now contains pattern.es, de, fr, it and nl modules for multilingual text analysis, with part-of-speech taggers for Spanish, German, French, Italian and Dutch, and sentiment analysis for Dutch and French (Italian is upcoming). The pattern.web module now supports CSS selectors that make parsing HTML trees more flexible and scalable. The pattern.vector module now comes bundled with LIBLINEAR for fast linear SVM's (Fan, Chang, Hsieh, Wang & Lin, 2008). Finally, our most recent addition is a pattern.server module that can be used to set up web services. It is based on CherryPy[1], and has syntax similar to Flask[2].

[1] http://www.cherrypy.org
[2] http://flask.pocoo.org

Pattern for Computational Creativity

Pattern does not specialise in any particular task. For each task, it provides one or two well-known approaches, usually one that is intuitive and one that is faster (e.g., k-NN vs. SVM). Users that need more may move on to specialised toolkits such as NLTK for natural language processing (Bird, Klein & Loper, 2009) and Scikit-learn for machine learning (Pedregosa, Varoquaux, Gramfort, Michel et al., 2011) as their projects become more involved.

Instead, Pattern offers creative leverage by allowing its users to freely combine a range of cross-domain tools. For example, the toolkit comes bundled with a common sense dataset, which can be traversed as a semantic network with pattern.graph to generate creative concepts (e.g., "Brussels, the toad", see De Smedt, 2013). The pattern.web module can then be used to search the web for evidence whether or not such concepts already exists to assess their novelty ("external validation", Veale, Seco & Hayes, 2009). Or, pattern.web can be used to mine words, word inflections and their parts-of-speech from the Italian Wiktionary, and analysed with the pattern.metrics helper module to construct an Italian part-of-speech tagger and regular expressions for Italian verb conjugation (De Smedt, Marfia, Matteucci & Daelemans, in press). With pattern.server we can subsequently launch a web service for Italian part-of-speech tagging that others can harness for language generation games, for example.

One user has compared Pattern to a "Swiss Army knife". Another user has called it a "treasure trove". In short, the toolkit is not designed for a specific purpose; rather it provides an open-ended range of tools that can be combined and explored – similar in philosophy to Boden's view on creativity (Boden, 2006). We think that Python coders who need to deal with data mining, natural language processing, machine learning, and so on, and who are active in the digital humanities and in the computational creativity (CC) community, will find Pattern useful, especially with its new pattern.server module.

Web Services with Pattern

Computational creativity covers a diverse range of tasks. It has been argued that web services are beneficial to the CC community (Veale, 2013). Different researchers can work on different tasks and share their results without having to reinvent algorithms from published pseudo code, deal with myriad installation instructions or adopt new programming languages. Instead, a request is sent to a web service and the response can be incorporated into any project. Many different web services can be combined to augment novel creativity research.

To demonstrate how web services work in Pattern, the example below implements a web service for semantic similarity, using just a few lines of code. Pattern comes bundled with WordNet 3 (Fellbaum, 1999). It also has an algorithm for Lin's semantic similarity (Lin, 1998), which measures the likelihood that two concepts occur in the same context, and whether they have a common ancestor in the WordNet graph. The `similarity()` function in this example takes two nouns, retrieves their WordNet synsets and estimates the semantic similarity between the two synsets as a value between 0.0 and 1.0. For example, the similarity between "cat" and "dog" is 0.86, whereas the similarity between "cat" and "teapot" is 0.0.

The `@app.route()` decorator defines the relative URL path where the web service is available. Optional keyword arguments of the `similarity()` function can be passed as URL query string parameters. The `similarity()` function returns a Python dictionary that will be served as a JSON-formatted string. Finally, the `app.run()` function starts the server.

```
from pattern.en import wordnet
from pattern.server import App

app = App()

@app.route('/similarity')
def similarity(w1='', w2=''):
 synset1 = wordnet.synsets(w1)[0]
 synset2 = wordnet.synsets(w2)[0]
 s = synset1.similarity(synset2)
 return {'similarity': round(s, 2)}

app.run('127.0.0.1', 8080, embedded=False)
```

In this case, the server runs locally. With `embedded=True` it will run as a mod_wsgi process on an Apache server. An optional parameter `debug=True` can be used to enable or disable error messages.

To try it out, we can execute the source code and visit http://127.0.0.1:8080/similarity?w1=cat&w2=dog in a web browser. The response is `{'similarity': 0.86}`. The example can be expanded with input validation and support for different word senses and word types.

Case Study: Weaseling Web Service

The following example demonstrates how pattern.en and pattern.server can be combined into a *weaseling* service for linguistic creativity. Weasel words are used to convey an air of meaningfulness in vague or ambiguous statements, as in "experts have claimed that this could be …".

The `weasel()` function takes a sentence and injects modal verbs so that, for example, "is" becomes "could be". The given sentence is part-of-speech tagged and verbs are transformed for common cases: non-action verbs get an additional "might" (e.g., "want" = "might want"), other verbs are passed to the pattern.en `conjugate()` function to transform them into the present participle tense (e.g., "run" = "might be running").

```
from pattern.en import parsetree
from pattern.en import conjugate
from pattern.server import App
from random import random

NONACTION = set((
   'appear', 'believe', 'contain', 'doubt',
   'exist', 'fear', 'feel', 'hate', 'hear',
   'hope', 'know', 'look', 'love', 'mean',
   'need', 'prefer', 'see', 'seem', 'sound',
   'think', 'understand', 'want', 'wish'
))

app = App()

@app.route('/weasel')
def weasel(s=''):
 r = []
 for sentence in parsetree(s, lemmata=True):
   for w in sentence:
     if r and w.tag.startswith('VB') \
      and random() < 0.05:
       r.append('often')
     if not w.tag.startswith('VB'):
       r.append(w.string.lower())
     elif w.lemma in ('be', 'have') \
      and w.tag not in ('VB', 'VBG', 'VBD'):
       r.append('might')
       r.append(w.lemma)
     elif w.lemma in ('be', 'have') \
      and w.tag == 'VBD':
       r.append('might')
       r.append('have')
       r.append(conjugate(w.lemma, 'VBN'))
     elif w.tag in ('VBP', 'VBZ') \
      and w.lemma in NONACTION:
       r.append('might')
       r.append(w.lemma)
     elif w.tag in ('VBP', 'VBZ'):
       r.append('might')
       r.append('be')
       r.append(conjugate(w.lemma, 'VBG'))
     else:
       r.append(w.string.lower())
   return ' '.join(r)

app.run('127.0.0.1', 8080, embedded=False)
```

For brevity, case sensitivity, punctuation, negation, verbs preceded by infinitival to, and verbs in the past tense are not handled. We can further improve the algorithm by injecting adverbs such as "often" and "perhaps" in a smarter way, transform quantifiers to vague expressions ("two" = "many"), and so on.

To try it out, we can execute the source code and visit http://127.0.0.1:8080/weasel?s=the+information+centre+is+to+the+north+of+here. The response is: "the information centre often might be to the north of here". Similarly, "you need a parking ticket" becomes "you might need a parking ticket", "this rental car runs on diesel fuel" becomes "this rental car might be running on diesel fuel" and "your hotel room was already paid for" becomes "your hotel room might have been already paid for".

The following code snippet queries our weaseling web service (running locally) and transforms Twitter statuses that contain a #travel hashtag:

```
from pattern.web import Twitter
from pattern.web import URL
from pattern.web import encode_url
from pattern.web import decode_utf8

API = 'http://127.0.0.1:8080/weasel?s='

for tweet in Twitter().search('#travel'):
  s = tweet.text
  r = URL(API + encode_url(s)).download()
  print decode_utf8(r)
  print
```

One tweet now states: "Miami International Airport often might be experiencing arrival delays of up to 30 minutes". Then again, it might not.

Further reading

De Smedt's doctoral dissertation[3] (2013) has more in-depth case studies of how Pattern can be used for CC.

For example, it discusses PERCOLATOR, a program that generates visuals based on today's news, FLOWEREWOLF, a poetry generator, PERCEPTION, a semantic network of commonsense, and MAD TEA PARTY, a problem solving algorithm (e.g., to open a locked door for which you don't have a key, you stubbornly club it with an albatross).

Future Work

Our new pattern.server module is not documented yet. Some examples of use are included in the latest Pattern release. We will provide extensive documentation[4] and unit tests once all lingering bugs have been fixed. Interested users are encouraged to contribute updates on GitHub[5].

Pattern is not ready yet for Python 3, unfortunately. Some preliminary steps have already been taken to make the toolkit available for Python 3. Work will continue along this line in the future.

Acknowledgements

The development of Pattern is supported by the Computational Linguistics Research Group at the University of Antwerp, Belgium, and the Experimental Media Research Group at the St Lucas University College of Art & Design, Antwerp, Belgium.

References

De Smedt, T., and Daelemans, W. 2012. Pattern for python. *The Journal of Machine Learning Research*, 13(1): 2063-2067.

Fan, R. E., Chang, K. W., Hsieh, C. J., Wang, X. R., and Lin, C. J. 2008. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research*, 9: 1871-1874.

Bird, S., Klein, E., and Loper, E. 2009. *Natural language processing with Python*. O'Reilly Media, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. 2011. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12: 2825-2830.

De Smedt, T., Marfia, F., Matteucci, M., Daelemans, W. In press. Using Wiktionary to build an Italian part-of-speech tagger. In *Proceedings of NLDB 2014*.

De Smedt, T. 2013. *Modeling Creativity: Case Studies in Python* (doctoral thesis). University Press Antwerp. ISBN 978-90-5718-260-0.

Veale, T., Seco, N., & Hayes, J. 2004. Creative discovery in lexical ontologies. In *Proceedings of the 20th international conference on Computational Linguistics*, 1333. Association for Computational Linguistics.

Boden, M. A. 2003. *The creative mind: Myths and mechanisms*. Routledge.

Fellbaum, C. 1999. *WordNet*. Blackwell Publishing Ltd.

Lin, D. 1998. An information-theoretic definition of similarity. In *ICML,* 98: 296-304.

Veale, T. 2013. A Service-Oriented Architecture for Computational Creativity. *Journal of Computing Science and Engineering, 7*(3): 159-167.

[3] http://bit.ly/modeling-creativity
[4] http://www.clips.ua.ac.be/pages/pattern-server
[5] http://www.github.com/clips/pattern