Get Started with NLP+AI in Python

Big Data Spain Madrid 2017-11-15

Paco Nathan @pacoid
O'Reilly Media

Daniel Vila Suero @dvilasuero
Recogn.ai

This course is for you because...

- You’re a Python programmer and need to learn how to use available NLP packages
- You’re a data scientist with some Python experience and need to leverage NLP and text mining
- You’re interested in chatbots, deep learning, and related AI work, and want to understand the basics for handling text data

Prerequisites

- Some programming in Python (we'll use Python 3)
- Basic understanding of HTML and the DOM structure for web pages
- Access to a computer with a browser

Optional: review **Probabilistic Data Structures in Python**

Preparation

To install the Python packages and work with the code on your laptop:

- Know how to install Python libraries using **PIP**, etc.
- Basic familiarity with Git and use of **GitHub**
- Also, it may be helpful to use **virtualenv**

You will need plus **Python 3**, plus:

- **Jupyter**
- **BeautifulSoup4**
- **TextBlob**
- **spaCy**
- **datasketch**
- **gensim**

Running a Jupyter notebook

Activate the Python environment, for example:

```
source ~/venv/bin/activate
```

Download the GitHub repo for this course:

https://github.com/ceteri/a41124835ed0

Connect into the downloaded repo and follow instructions in the **`README.md`** file

Make sure you have Jupyter installed:

jupyter.org/install.html

Then launch Juypter:

```
cd a41124835ed0
jupyter notebook
```

github.com/ceteri/a41124835ed0

Common misunderstandings

- How keyword analysis, n-grams, co-occurrence, stemming, and other techniques from a previous generation of NLP tools are no longer the best approaches to use
- That NLP work leading into AI applications is either fully automated or something which requires a huge amount of manual work
- NLP requires Big Data tooling and use of clusters
- ML/NLP/AI solutions in Python do not scale

Expected outcomes

Participants will understand…

- Benefits of using Python for NLP applications
- How statistical parsing works
- How resources such as WordNet enhance text mining
- How to extract more than just a list of keywords from a text
- How to summarize and compare a set of documents
- How deep learning gets used with natural language

Expected outcomes

Participants will be able to…

- Prepare texts for *parsing*, e.g., how to handle difficult Unicode
- Parse sentences into *annotated* lists, structured as JSON output
- Perform *keyword ranking* using TF-IDF, while filtering stop words
- Calculate a *Jaccard similarity* measure to compare texts
- Leverage *probabilistic data structures* to perform the above more efficiently
- Use *Jupyter notebooks* for sharing results within their teams

Section 1:
Getting the text…

30 min

Why use Python for NLP?

Python provides a number of excellent packages for natural language processing (NLP) along with great ways to leverage the results.

If you're new to NLP, this course provides initial hands-on work: confidence to explore further into **Deep Learning**, natural language generation, **chatbots**, etc.

First we'll show how to prepare text for parsing, how to extract key phrases, prepare text for indexing in search, calculate similarity between documents, etc.

Extracting text from HTML

Suppose we have an HTML document organized like the following:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Foo Bar</title>
</head>
<body>
<div id="article-body">
<p>Nullum gratuitum prandium. Phasellus dictum urna sed metus
aliquet, quis vehicula ex rhoncus. In ante urna, imperdiet in
placerat non, elementum a libero.</p>
<p>Nam eu sem metus. Interdum et malesuada fames ac ante ipsum
primis in faucibus. Quisque et hendrerit massa.</p>
</div>
<div id="boiler-plate">
<p>Copyright ©2015 Fuberz. All rights reserved.</p>
</div>
</body>
</html>
```

Extracting text from HTML

We want the text in the **<p/>** elements enclosed by the **<div/>** element with **id="article-body"**

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Foo Bar</title>
</head>
<body>
<div id="article-body">
<p>Nullum gratuitum prandium. Phasellus dictum urna sed metus
aliquet, quis vehicula ex rhoncus. In ante urna, imperdiet in
placerat non, elementum a libero.</p>
<p>Nam eu sem metus. Interdum et malesuada fames ac ante ipsum
primis in faucibus. Quisque et hendrerit massa.</p>
</div>
<div id="boiler-plate">
<p>Copyright ©2015 Fuberz. All rights reserved.</p>
</div>
</body>
</html>
```

Extracting text from HTML

We'll use a popular library called **Beautiful Soup** to extract text from HTML.

Run Jupyter in the directory for the course repo, open the **`ex01.ipynb`** notebook, then run its code examples:

- open an example HTML document
- select the **`<div/>`** that has the article content
- iterate through the **`<p/>`** paragraphs
- extract their text

Extracting text from HTML

Using the first HTML article, you should see text results starting with:

```
Almost a year ago, we published our now-
annual landscape of machine intelligence
companies
```

Try some of the other articles and re-run those code blocks.

Concerns about character encoding

The text in the HTML documents that we've been using is relatively "clean" … though that rarely happens in practice!

- "**Text vs. Data Instead of Unicode vs. 8-bit**"
- **ligatures** used in publishing add to the fun
- **codecs** become especially important for storing text in files
- See also about **Unicode equivalence** and how to **normalize**

Open the **`ex02.ipynb`** notebook, then run its code examples.

Concerns about character encoding

Character encoding is a hard problem, and will continue to be a hard problem – along with annotations. Don't hold your breath for either to become automated soon. But it could happen.

For more details and further context about use cases in search, check the excellent article "**Character Filtering**" by Daniel Tunkelang, along with his entire series about **Query Understanding**.

Related work is important in recommender systems, customer support, chatbots, etc.

Applications where NLP matters

Increasingly, customers send text to interact or leave comments, which provides a wealth of data for text mining.

That’s a great starting point for developing custom search, content recommenders, and even **AI applications**.

Section 2:
Statistical parsing and annotation

40 min

Statistical parsing

NLP used to be mostly concerned about **transformational grammars**, linguistic theory by Chomsky, etc. Other approaches such as **link grammars** are largely overlooked.

ML techniques allow much simpler approaches called *statistical parsing*. With the rise of Big Data, these techniques became even more effective – **Google won the NIST 2005 Machine Translation Evaluation** and remains a leader. Another notable project is the **Stanford Parser**.

Probabilistic methods split texts into sentences, annotate words with part-of-speech, chunk noun phrases, resolve named entities, estimate sentiment scores, etc.

Intro to using spaCy, TextBlob, WordNet, etc.

We'll use a few popular NLP resources for parsing text:

- **spaCy** – one of the top NLP libraries in Python
- **TextBlob** – a Python 2/3 library that provides a consistent API for leveraging other resources
- **WordNet** – think of it as somewhere between a large thesaurus and a database

One important step is to annotate the words in each sentence with a tag that describes its part of speech: noun, verb, adjective, determinant, adverb, etc.

Intro to using spaCy, TextBlob, WordNet, etc.

```
text = '''
The titular threat of The Blob has always struck me as the ultimate movie
monster: an insatiably hungry, amoeba-like mass able to penetrate
virtually any safeguard, capable of--as a doomed doctor chillingly
describes it--"assimilating flesh on contact.
Snide comparisons to gelatin be damned, it's a concept with the most
devastating of potential consequences, not unlike the grey goo scenario
proposed by technological theorists fearful of
artificial intelligence run rampant.
'''

from textblob import TextBlob
blob = TextBlob(text)
print(blob.tags)
print(blob.noun_phrases)
```

TextBlob

Based on using `TextBlob` for natural language processing in Python: https://textblob.readthedocs.io/en/dev/

In [2]:
```
text = '''
The titular threat of The Blob has always struck me as the ultimate movie
monster: an insatiably hungry, amoeba-like mass able to penetrate
virtually any safeguard, capable of--as a doomed doctor chillingly
describes it--"assimilating flesh on contact.
Snide comparisons to gelatin be damned, it's a concept with the most
devastating of potential consequences, not unlike the grey goo scenario
proposed by technological theorists fearful of
artificial intelligence run rampant.
'''
```

In [3]:
```
from textblob import TextBlob
blob = TextBlob(text)
```

In [4]:
```
print(blob.tags)
```

```
[('The', 'DT'), ('titular', 'JJ'), ('threat', 'NN'), ('of', 'IN'), ('The', 'DT'), ('Blob', 'NNP'), ('has', 'VBZ'),
 ('always', 'RB'), ('struck', 'VBN'), ('me', 'PRP'), ('as', 'IN'), ('the', 'DT'), ('ultimate', 'JJ'), ('movie', 'N
N'), ('monster', 'NN'), ('an', 'DT'), ('insatiably', 'RB'), ('hungry', 'JJ'), ('amoeba-like', 'JJ'), ('mass', 'NN'),
 ('able', 'JJ'), ('to', 'TO'), ('penetrate', 'VB'), ('virtually', 'RB'), ('any', 'DT'), ('safeguard', 'NN'), ('capabl
e', 'JJ'), ('of', 'IN'), ('as', 'IN'), ('a', 'DT'), ('doomed', 'JJ'), ('doctor', 'NN'), ('chillingly', 'RB'), ('descr
ibes', 'VBZ'), ('it', 'PRP'), ('assimilating', 'VBG'), ('flesh', 'NN'), ('on', 'IN'), ('contact', 'NN'), ('Snide', 'J
J'), ('comparisons', 'NNS'), ('to', 'TO'), ('gelatin', 'VB'), ('be', 'VB'), ('damned', 'VBN'), ('it', 'PRP'), ("'s",
 'VBZ'), ('a', 'DT'), ('concept', 'NN'), ('with', 'IN'), ('the', 'DT'), ('most', 'RBS'), ('devastating', 'JJ'), ('o
f', 'IN'), ('potential', 'JJ'), ('consequences', 'NNS'), ('not', 'RB'), ('unlike', 'IN'), ('the', 'DT'), ('grey', 'N
N'), ('goo', 'NN'), ('scenario', 'NN'), ('proposed', 'VBN'), ('by', 'IN'), ('technological', 'JJ'), ('theorists', 'NN
S'), ('fearful', 'NN'), ('of', 'IN'), ('artificial', 'JJ'), ('intelligence', 'NN'), ('run', 'NN'), ('rampant', 'NN')]
```

In [5]:
```
print(blob.noun_phrases)
```

```
['titular threat', 'blob', 'ultimate movie monster', 'amoeba-like mass', 'snide', 'potential consequences', 'grey goo
 scenario', 'technological theorists fearful', 'artificial intelligence run rampant']
```

If you did not have `TextBlob` installed before you will get exceptions about `nltk.download()` and downloading some required data sets. See http://www.nltk.org/data.html

Lemmatization vs. Stemming

Use of *stemming*, e.g., with **Porter Stemmer**, has long been a standard way to “normalize” text data: a computationally efficient approach to reduce words to their “stems” by removing inflections.

A better approach is to *lemmatize*, i.e., use part-of-speech tags to lookup the root for a word in WordNet – related to looking up its *synsets*, *hypernyms*, *hyponyms*, etc.

Lexeme	PoS	Stem	Lemma
interact	VB	interact	interact
comments	NNS	comment	comment
provides	VBZ	provid	provide

Splitting sentences and PoS annotation

In the next exercise, we'll parse sentences using spaCy, then use TextBlob and WordNet to get more info about each word:

- clean-up text data
- split sentences
- annotate with part-of-speech (PoS) tags
- lemmatize nouns and verbs
- lookup synsets and definitions

Open the **`ex03.ipynb`** notebook, then run its code examples.

Noun phrase chunking

Consider the sentence:

```
A great starting point for developing custom search.
```

There's much more information in the key phrase "`custom search`" than there is in the individual keywords "`custom`" and "`search`".

We can use spaCy to perform *noun phrase chunking* to extract the noun phrases. It's not perfect, but sometimes quite helpful.

Open the **`ex04.ipynb`** notebook, then run its code examples.

Named entity resolution

A special case of noun phrases involves *proper nouns*, and for that we can use an approach called *named-entity resolution*.

Consider a scenario using NLP to augment customer service interaction for an online shopping catalog. You'd probably need to identify most of the product names and their popular abbreviations as named entities. You may even need to include common misspellings. This shifts quickly into *natural language understanding*.

Open the **`ex05.ipynb`** notebook, then run its code examples. *NB: bugs on spaCy web site examples.*

Store annotated text as JSON files

Now that we have some NLP code for parsing texts, it'll easier to use if we create a small library and then call functions from it. Take a look at the Python source code in **`pynlp.py`**

The function **`pynlp.full_parse()`** extracts text from HTML, then stores the parsed and annotated text as JSON, one line per sentence.

Open the **`ex06.ipynb`** notebook, then run its code examples. In the empty code block at the end, write Python code to run the extract/parse/save-to-JSON for each of the example HTML files in the course repo.

Section 3:
Fun things to do with annotated text

30 min

TF-IDF for ranking terms

TF-IDF is an acronym for *term frequency - inverse document frequency*. This metric gets used to weight the keywords or key phrases found in a document, to create *feature vectors*.

The **`tf`** portion considers how frequently a term appears in a document, normalized by the **`df`** portion which considers how frequently terms appear across all documents.

Open the **`ex07.ipynb`** notebook, then run its code examples.

NB: there are many variants for how to calculate this metric; we're using:

$$IDF(t, D) = \log \frac{|D| + 1}{DF(t, D) + 1}$$

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D)$$

Semantic similarity

Semantic similarity between two texts can be measured using a variety of approaches. **Jaccard and Tanimoto** are two statistical measures for *sample sets* of features.

We'll approximate a Jaccard similarity with **MinHash** – the **Probabilistic Data Structures in Python** tutorial has more detailed background about this.

This is particularly good if you want to construct a graph to describe how text documents are related, e.g., for building a *content recommender*.

Open the **`ex08.ipynb`** notebook, then run its code examples.

TextRank to extract key phrases

TextRank was introduced in 2004 by Rada Mihalcea and Paul Tarau, as a way to extract key phrases for auto-summarization – which improved greatly over use of single keywords, n-grams, etc.

The gist is:

1. construct a graph from a paragraph of text
2. run **PageRank** on that graph
3. extract the highly ranked noun phrases

For a pure-Python implementation, take a look at **pytextrank** which we'll demo here:

- **https://github.com/ceteri/pytextrank**
- **https://pypi.python.org/pypi/pytextrank/**

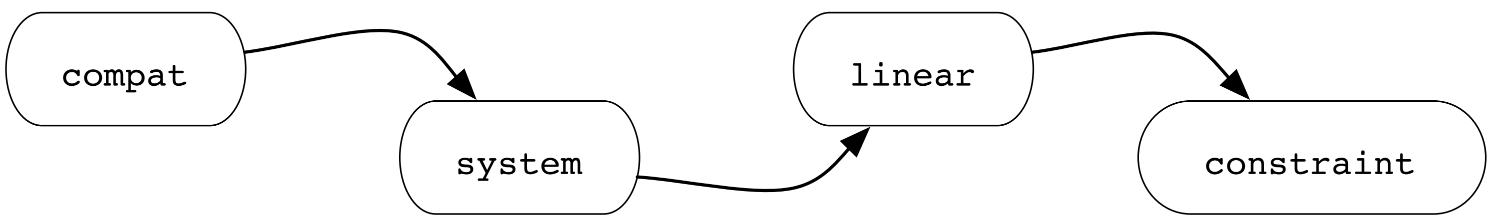
TextRank to extract key phrases

1: "Compatibility of systems of linear constraints"

2:

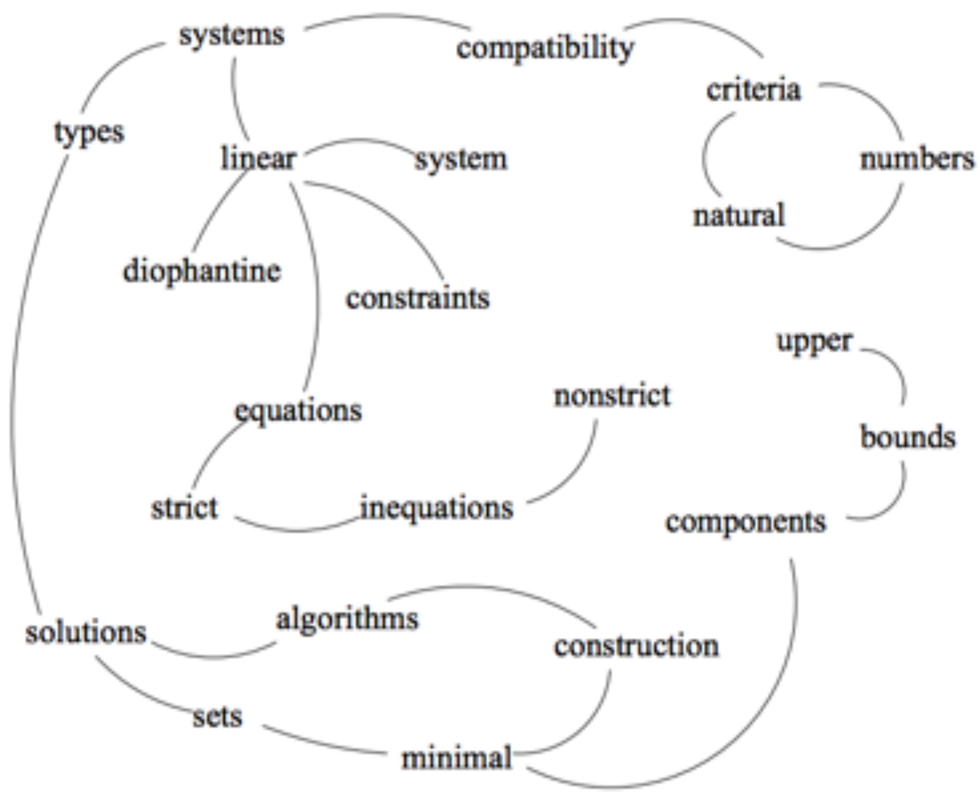
```
[{'index': 0, 'root': 'compatibility', 'tag': 'NNP','word': 'compatibility'},
 {'index': 1, 'root': 'of', 'tag': 'IN', 'word': 'of'},
 {'index': 2, 'root': 'system', 'tag': 'NNS', 'word': 'systems'},
 {'index': 3, 'root': 'of', 'tag': 'IN', 'word': 'of'},
 {'index': 4, 'root': 'linear', 'tag': 'JJ', 'word': 'linear'},
 {'index': 5, 'root': 'constraint', 'tag': 'NNS','word': 'constraints'}]
```

3:



TextRank to extract key phrases

Compatibility of systems of linear constraints over the set of natural numbers. Criteria of compatibility of a system of linear Diophantine equations, strict inequations, and nonstrict inequations are considered. Upper bounds for components of a minimal set of solutions and algorithms of construction of minimal generating sets of solutions for all types of systems are given. These criteria and the corresponding algorithms for constructing a minimal supporting set of solutions can be used in solving all the considered types systems and systems of mixed types.



Section 4:
A preview of advanced topics

20 min

Summarization

Historically, there are a few different ways to handle text **summarization**, including use of deep learning.

In this demo, we'll show a simple approach:

1. parse the text to create a feature vector
2. calculate the semantic similarity between the feature vector and each sentence
3. select the N top-ranked sentences, listed in order, to summarize

Almost a year ago, we published our now-annual landscape of machine intelligence companies, and goodness have we seen a lot of activity since then. As has been the case for the last couple of years, our fund still obsesses over 'problem first' machine intelligence -- we've invested in 35 machine intelligence companies solving 35 meaningful problems in areas from security to recruiting to software development. At the same time, the hype around machine intelligence methods continues to grow: the words 'deep learning' now equally represent a series of meaningful breakthroughs (wonderful) but also a hyped phrase like 'big data' (not so good!). What's the biggest change in the last year?

Vector embedding

Vector embedding methods map words, phrases, sentences, etc., to numerical vectors – generally trained using deep learning.

We'll review a demo based on books from Safari, using **Word2Vec** with the **gensim** library. Then we'll query to find related terms.

This approach can be used to enhance search. Take a look at **GPU Accelerated Natural Language Processing** by Guillermo Moliní, which describes the **Happening** platform for semantic search.

Vector embedding

```
import gensim

# train a model, then save it
model = gensim.models.Word2Vec(sentences, min_count=1)
model.save(MODEL_FILE)

# ---

# load and query a trained model
model = gensim.models.Word2Vec.load(MODEL_FILE)

while True:
  query = input("\nquery? ")
  get_synset(model, query)
```

Using LSTM to generate film scripts

Long short-term memory (LSTM) is an approach that allows recurrent neural networks to learn over many time steps. These can be used for time-series analysis, and also for learning sequences of data, such as in streams of voice or text.

Imagine feeding many scripts (semi-structured text) from a particular film genre through an LSTM, then generating new output. That's what **Benjamin.wtf** does…

Using LSTM to generate film scripts

Sunspring
youtu.be/LY7x2Ihqjmc

Benjamin.wtf

It's No Game
goo.gl/Waw5Px

Using LSTM to generate music

https://github.com/IraKorshunova/folk-rnn

…even romance novels

Recommended by Quincy Larson, deb siegel, and 15 others

Elle O'Brien
Computational scientist, software developer, science writer
Aug 6 · 6 min read

Romance Novels, Generated by Artificial Intelligence

I've always been fascinated with romance novels—the kind they sell at the drugstore for a couple of dollars, usually with some attractive, soft-lit couples on the cover. So when I started futzing around with text-generating neural networks a few weeks ago, I developed an urgent curiosity to discover what artificial intelligence could contribute to the ever-popular genre. Maybe one day there will be entire books written by computers. For now, let's start with titles.

Recommended resources

Get Started with Natural Language Processing Using Python, Spark, and Scala

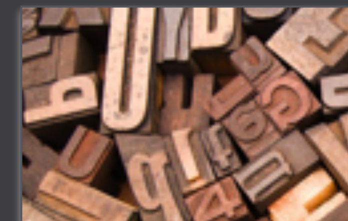
Mastering SpaCy for Natural Language Processing

Text Mining & Natural Language Understanding at Scale

Artificial Intelligence: Teaching Machines to Think Like People

O’Reilly Media conferences + training:

NLP in Python
repeated live online courses



Strata Data Conference
SG, Dec 4-7
SJ, Mar 5-8
UK, May 21-24
CN, Jul 12-15
NY, Sep 11-14

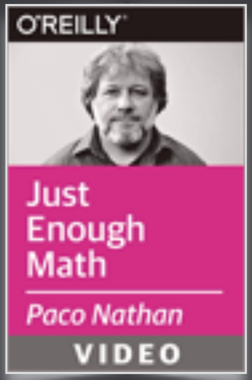
The AI Conf
CN, Apr 11-13
NY, Apr 29-May 2
SF, Sep 4-7
UK, Oct 8-11

JupyterCon
NY, Aug 21-24

instructor:

updates, reviews, conference summaries, etc.:

liber118.com/pxn/
@pacoid



Just Enough Math

Building Data Science Teams

Learn Alongside Innovators

Hylbert-Speys

How Do You Learn?