Natural Language Processing using PYTHON
(with NLTK, scikit-learn and Stanford NLP APIs)

Instructor: Diptesh Kanojia, Abhijit Mishra
Supervisor: Prof. Pushpak Bhattacharyya
Center for Indian Language Technology
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
email: {diptesh,abhijitmishra,pb}@cse.iitb.ac.in
URL: http://www.cse.iitb.ac.in/~{diptesh,abhijitmishra,pb}
**Roadmap**

- **Session 1 (Introduction to NLP, Shallow Parsing and Deep Parsing)**
  - Introduction to python and NLTK
  - Text Tokenization, POS tagging and chunking using NLTK.
  - Constituency and Dependency Parsing using NLTK and Stanford Parser

- **Session 2 (Named Entity Recognition, Coreference Resolution)**
  - NER using NLTK
  - Coreference Resolution using NLTK and Stanford CoreNLP tool

- **Session 3 (Meaning Extraction, Deep Learning)**
  - WordNets and WordNet-API
  - Other Lexical Knowledge Networks – Verbnet and Framenet
SESSION-1 (INTRODUCTION TO NLP, SHALLOW PARSING AND DEEP PARSING)

- Introduction to python and NLTK
- Text Tokenization, Morphological Analysis, POS tagging and chunking using NLTK.
- Constituency and Dependency Parsing using NLTK

Expected duration: 15 mins
**Why Python?**

As far as I know, you cannot execute a Python program (compiled to bytecode) on every machine, such as on windows, or on linux without modification.

You are incorrect. The python bytecode is cross platform. See Is python bytecode version-dependent? Is it platform-dependent? on Stack Overflow.

However, it is not compatible across versions. Python 2.6 cannot execute Python 2.5 files. So while cross-platform, its not generally useful as a distribution format.

**But why Python needs both a compiler and an interpreter?**

Speed. Strict interpretation is slow. Virtually every "interpreted" language actually compiles the source code into some sort of internal representation so that it doesn't have to repeatedly parse the code. In python's case it saves this internal representation to disk so that it can skip the parsing/compiling process next time it needs the code.
Introduction to python

- A programming language with strong similarities Perl and C with powerful typing and object oriented features.
  - Commonly used for producing HTML content on websites. (e.g. Instagram, Bitbucket, Mozilla and many more websites built on python-django framework).
  - Great for text processing (e.g. Powerful RegEx tools).
  - Useful built-in types (lists, dictionaries, generators, iterators).
  - Parallel computing (Multi-processing, Multi-threading APIs)
  - Map-Reduce facilities, lambda functions
  - Clean/Readable syntax, Lots of open-source standard libraries
  - Code Reusability

- DEMO (basics.py)
Installing Python

- Download and installation instructions at:
  - https://www.python.org/download/

- Windows/Mac systems require installation

- Pre-installed latest Linux distributions (Ubuntu, Fedora, Suse etc.)

- We will use Python 2.7.X versions.
Python Tutorials

- “Dive into Python”
  http://diveintopython.org/

- The Official Python Tutorial
  https://docs.python.org/2/tutorial/

- The Python Quick Reference
  http://rgruet.free.fr/PQR2.3.html
Useful IDE/Text-Editors

- IDLE (Windows)
- Vi/Emacs (Linux)
- Geany (Windows/Linux/Mac)
- Pydev plugin for Eclipse IDE (Windows/Linux/Mac)
- Notepad++ (Windows)
Useful Python Libraries

- **NumPy** (Mathematical Computing, Advanced mathematical functionalities)
- **Matplotlib** (Numerical plotting library, useful in data analysis)
- **Scipy** (Library for scientific computation)
- **Scikit-learn** (Machine Learning/Data-mining library, 
- **PIL** (Python library for Image Processing)
- **PySpeech** (Library for speech processing and text-to-speech conversion)
- **XML/LXML** (XML Parsing and Processing)
- **NLTK** (Natural Language Processing)
  And many more…

https://wiki.python.org/moin/UsefulModules
The Natural Language Toolkit (NLTK)

- Developed by Steven Bird and Co. at Stanford University (2006).
- Open source python modules, datasets and tutorials
- Papers:
Components of NLTK (Bird et. al, 2006)

1. **Code**: corpus readers, tokenizers, stemmers, taggers, chunkers, parsers, wordnet, ... (50k lines of code)
2. ** Corpora**: >30 annotated data sets widely used in natural language processing (>300Mb data)
3. **Documentation**: a 400-page book, articles, reviews, API documentation
1. Code

- Corpus Readers
- Tokenizers
- Stemmers
- Taggers
- Parsers
- WordNet
- Semantic Interpretation
- Clusterers
- Evaluation Metrics
2. Corpora

- **Brown Corpus**
- **Carnegie Mellon Pronouncing Dictionary**
- CoNLL 2000 Chunking Corpus
- Project Gutenberg Selections
- NIST 1999 Information Extraction: Entity Recognition Corpus
- US Presidential Inaugural Address Corpus
- **Indian Language POS-Tagged Corpus**
- Floresta Portuguese Treebank
- Prepositional Phrase Attachment Corpus
- **SENSEVAL 2 Corpus**
- Sinica Treebank Corpus Sample
- Universal Declaration of Human Rights Corpus
- **Stopwords Corpus**
- TIMIT Corpus Sample
- Treebank Corpus Sample
3. Documentation

- **Books:**
  - Natural Language Processing with Python - Steven Bird, Edward Loper, Ewan Klein
  - Python Text Processing with NLTK 2.0 Cookbook – Jacob Perkins

- **Included in NLTK:**
  - Installation instructions
  - API Documentation: describes every module, interface, class, and method
NLTK - How to?

- Install NLTK
  - Follow instructions at http://www.nltk.org/install.html
  - Installers for Windows, Linux and Mac OS available

- Check installation
  - Execute python command through “shell” (Linux/Mac) or command prompt “cmd” (Windows).
    - $python
    - >>import nltk
  - The interpreter should import “nltk” without showing any error.

- Download NLTK data (corpora):
  - >>nltk.download()
# NLTK modules

<table>
<thead>
<tr>
<th>NLP Tasks</th>
<th>NLTK Modules</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessing Corpora</td>
<td>nltk.corpus</td>
<td>Standardized interfaces to corpora and lexicons</td>
</tr>
<tr>
<td>String Processing</td>
<td>nltk.tokenize, nltk.stem</td>
<td>Tokenizers, sentence tokenizers, stemmers</td>
</tr>
<tr>
<td>Collection Discovery</td>
<td>nltk.collections</td>
<td>t-test, chi-squared, point-wise mutual information</td>
</tr>
<tr>
<td>POS Tagging</td>
<td>nltk.tag</td>
<td>n-gram, backoff, Brill, HMM, TnT</td>
</tr>
<tr>
<td>Chunking</td>
<td>nltk.chunk</td>
<td>Regular expression, n-gram, named entity</td>
</tr>
<tr>
<td>Parsing</td>
<td>nltk.parse</td>
<td>Chart, feature-based, unification, probabilistic, dependency</td>
</tr>
<tr>
<td>Classification</td>
<td>nltk.classify, nltk.cluster</td>
<td>Decision tree, maximum entropy, naive Bayes, EM, k-means</td>
</tr>
<tr>
<td>Semantic Interpretation</td>
<td>nltk.sem, nltk.inference</td>
<td>Lambda calculus, first-order logic, model checking</td>
</tr>
<tr>
<td>Evaluation Metrics</td>
<td>nltk.metrics</td>
<td>Precision, recall, agreement coefficients</td>
</tr>
<tr>
<td>Probability Estimation</td>
<td>nltk.probability</td>
<td>Frequency distributions, smoothed probability distributions</td>
</tr>
<tr>
<td>Applications</td>
<td>nltk.app</td>
<td>Graphical concordancer, parsers, WordNet browser</td>
</tr>
<tr>
<td>Linguistics fieldwork</td>
<td>nltk.toolbox</td>
<td>Manipulate data in SIL Toolbox format</td>
</tr>
</tbody>
</table>
Text Tokenization

- Process of splitting a string into a list of tokens (words, punctuation-marks etc.)
- For most of the languages whitespace separates two adjacent words.
- Exceptions (source: Wikipedia):
  - For Chinese and Japanese, sentences are delimited but words are not.
  - For Thai, Phrases and Sentences are delimited but not words.
Text Tokenization – NLTK Tokenizers

- Description: [http://www.nltk.org/api/nltk.tokenize.html](http://www.nltk.org/api/nltk.tokenize.html)
- Demo:
  - LineTokenizer — Tokenize string into lines
  - /*PunctWordTokenizer (Statistical) —
    - Tokenizing based on Unsupervised ML algorithms.
    - Model parameters are learnt through training on a large corpus of abbreviation words, collocations, and words that start sentences*/
  - RegexpTokenizer- Tokenization based on RegExp
  - SExprTokenizer — To find parenthesized expressions in a string
  - TreebankWordTokenizer — Tokenization as per Penn-Treebank standards
NLTK Morphological Analyzers

- **DEMO- Stemmers**
  - Lancaster Stemmer
  - Porter Stemmer
  - Regexp Stemmer
  - Snowball Stemmer

- **DEMO - Lemmatizers**
  - WordNet based Lemmatizers

Script: morphological_analyzer.py
NLTK - Part of Speech Tagging

- The process of sequentially labeling words in a sentence with their corresponding part of speech tags.

- Demo – NLTK POS Taggers (pos_tagger.py)
  - Unigram Tagger (Based on prior probability)
  - Brill Tagger (Rule Based)
  - Regexp Tagger (Using regular expressions for tagging)
  - HMM based Tagger (HMM-Viterbi based)
  - Stanford tagger (Using Stanford module)
  - NLTK recommended tagger
Parsers

- Writing grammar
- Rule based constituency parsing
  - RecursiveDescent Parser
  - ShiftReduce Parser
- DEMO- Statistical Parsers
  - Probabilistic Context Free Grammar (PCFG)
    - Stanford parser
  - Probabilistic Dependency Parsing
    - Malt Parser
    - Stanford Parser
- Script: parser_demo.py
SESSION 2 (NAMED ENTITY RECOGNITION, COREFERENCE RESOLUTION)

- NER using NLTK

Expected duration: 15 mins
NER and Coreference Resolution using NLTK

- DEMO – Named entity chunking using NLTK
- Using Stanford CoreNLP tool for Coreference Resolution
  - Download and installation instruction at [http://stanfordnlp.github.io/CoreNLP/](http://stanfordnlp.github.io/CoreNLP/)
  - Python Wrapper for CoreNLP
    [https://github.com/dasmith/stanford-corenlp-python](https://github.com/dasmith/stanford-corenlp-python)
  - DEMO – Coreference Resolution using Stanford CoreNLP
  - Scripts: coreference_resolution.py
    named_entity_chunking.py
SESSION-4 (MEANING EXTRACTION, DEEP LEARNING)

• WordNets and WordNet-API
• Other Lexical Knowledge Networks – VerbNet and FrameNet

Expected duration: 10 mins
WordNet

- DEMO - NLTK WordNet (wordnet.py)
  - Finding all the synonym set (SynSet) of a word for all possible pos tags.
  - Finding all the SynSets if POS tag is known.
  - Finding hypernym, hyponym of a synset.
  - Finding similarities between two words.
Other Lexical Networks – ConceptNet, VerbNet and FrameNet

- DEMO- Conceptnet (conceptnet.py)
  - Using Divisi API (since it is not available in NLTK)
  - Using ConceptNet for finding attributes of a concept.
  - Using ConceptNet for word similarity computation

- DEMO- VerbNet (framenet_verbnet.py)
  - Obtaining verb classes from VerbNet

- DEMO – FrameNet
  - Listing all the frames in FrameNet
  - Obtaining properties of a particular frame.
THANK YOU