Introduction to Programming with Python

Dr. Anatol Wegner

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

anatol.wegner@uni-wuerzburg.de

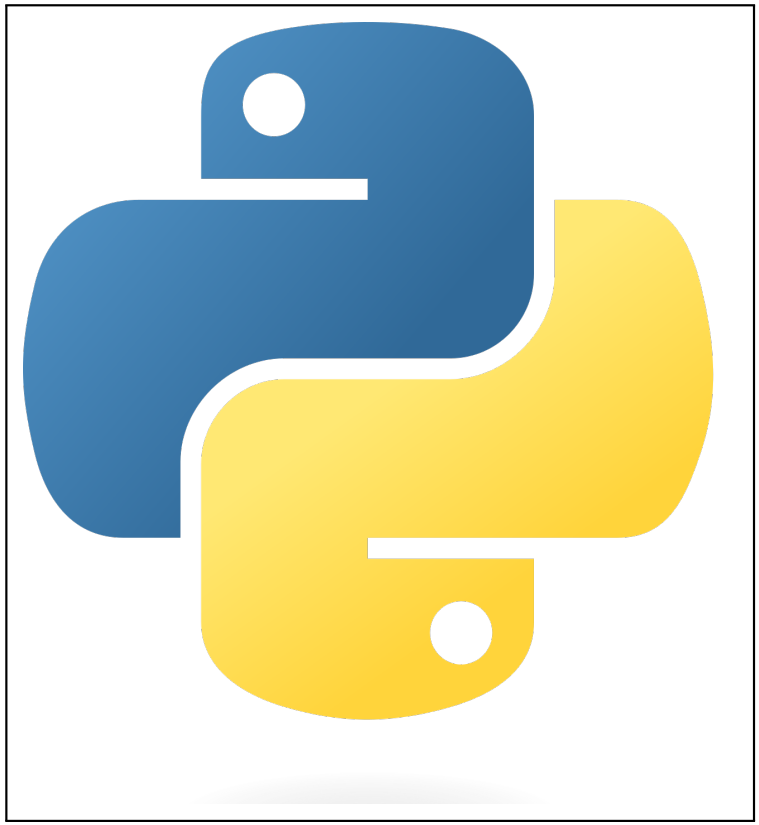
Lecture 07
Natural language processing

December 13, 2024

ML 4
MACHINE LEARNING FOR COMPLEX NETWORKS
NETS

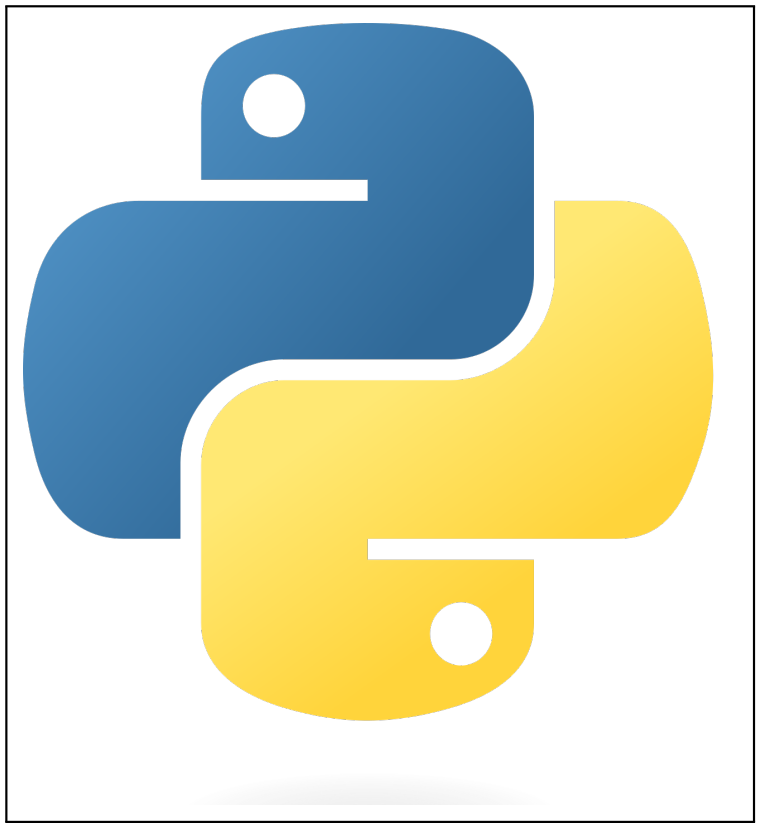
Recap

- **Supervised Learning**:
  - **Definition**: Models learn from labeled data to predict outputs for new inputs.
  - **Key Concepts**:
    - Classification (e.g., MNIST digit classification).
    - Algorithms: `KNeighborsClassifier`, SVC.
    - Train-Test Split: Preparing data for training and evaluation.



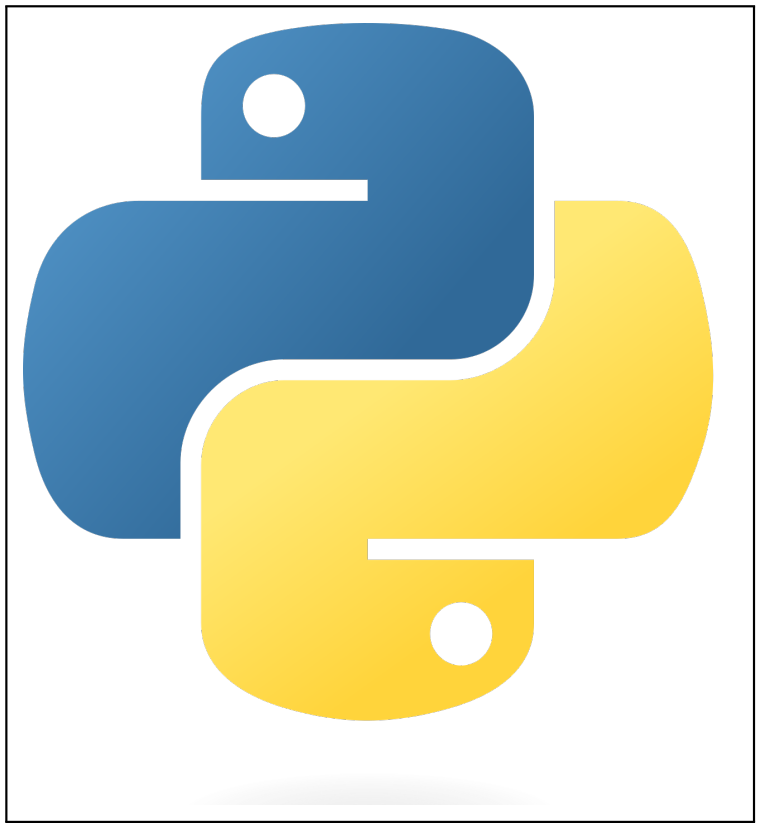
Recap

- **Unsupervised Learning**:
  - **Definition**: Models learn patterns and structures from unlabeled data.
  - **Key Concepts**:
    - Clustering (e.g., DBSCAN, K-Means on MNIST).
- **Scikit-learn Basics**:
  - Unified interface for training (`fit`) and predictions (`predict`).
  - Tools for preprocessing, model evaluation, and visualization.



Recap

- **Unsupervised Learning**:
  - **Definition**: Models learn patterns and structures from unlabeled data.
  - **Key Concepts**:
    - Clustering (e.g., DBSCAN, K-Means on MNIST).
- **Scikit-learn Basics**:
  - Unified interface for training (`fit`) and predictions (`predict`).
  - Tools for preprocessing, model evaluation, and visualization.



Today: Natural language processing

- Loading and preprocessing natural language data.
- Tokenization, part-of-speech tagging, lemmatization.
- Sentiment analysis.

What is NLP?

- **Definition:** Natural Language Processing (NLP) is a field of computer science that focuses on the computational analysis, modeling, and manipulation of natural language data.
- **Applications:**
  - Sentiment Analysis
  - Translation (e.g., Google Translate)
  - Search Engines
  - Chatbots and Virtual Assistants
- **Relevance to research:**
  - Analyzing historical documents
  - Understanding social media trends
  - Text mining in literature and archives

Introduction to NLTK

- **NLTK (Natural Language Toolkit):**
  - A comprehensive library for working with human language data (text).
  - Provides tools for:
    - Tokenization (splitting text into words or sentences).
    - Stemming and Lemmatization.
    - Part-of-Speech (POS) Tagging.
    - Parsing and Chunking.
    - Working with corpora (e.g., Gutenberg, WordNet).
  - Widely used in education and research.
- **Documentation:** https://www.nltk.org/

Loading Text Data

- Text data can come from various sources:
  - Local files (e.g., `.txt`, `.csv`).
  - Online resources or APIs.
  - Built-in datasets from libraries like NLTK.
- Importance of loading data correctly:
  - Ensures compatibility with preprocessing tools.
  - Facilitates efficient analysis.

Example: NLTK's Built-in Corpus

- NLTK provides several built-in text datasets:
  - Gutenberg corpus: Classic literature texts.
  - Movie reviews: Sentiment classification dataset.
- Text data in NLTK is represented as raw strings, tokenized lists, or tagged formats depending on the use case.
- Example: Loading and inspecting raw text from the Gutenberg corpus.

Python Example:

```
from nltk.corpus import gutenberg
print(gutenberg.fileids())
# Raw text as a single string:
text = gutenberg.raw('austen-emma.txt')
print(text[:500])  # First 500 characters of the text
```

Cleaning Text Data

- Raw text often needs cleaning before analysis:
  - Convert text to lowercase for uniformity.
  - Remove punctuation and special characters.
  - Handle whitespace and remove stopwords if necessary.
- Why is cleaning important?
  - Reduces noise in the data.
  - Improves accuracy of NLP tasks.

Python Example:

```
cleaned_text = text.lower() # Convert to lowercase

# Remove punctuation
punctuations = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
cleaned_text = cleaned_text.translate(
str.maketrans('', '', punctuations))
print(cleaned_text[:500])  # First 500 cleaned characters
```

What is Tokenization and Why is it Important?

- **Definition:** Tokenization is the process of breaking text into smaller units, called **tokens**.
- Tokens can be:
  - Words (word-level tokenization).
  - Sentences (sentence-level tokenization).
  - Characters (character-level tokenization).
- Tokenization is a crucial preprocessing step in NLP tasks such as:
  - Text classification.
  - Machine translation.
  - Sentiment analysis.
- Why Tokenize?
  - Provides structure to raw text by splitting it into analyzable units.
  - Simplifies tasks like frequency analysis, parsing, and syntactic analysis.
  - Helps handle multilingual text with different delimiters.

Tokenization in NLTK

- NLTK provides built-in functions for tokenization:
  - `word_tokenize` for word-level tokenization.
  - `sent_tokenize` for sentence-level tokenization.
- Example: Tokenizing text into words.

Python Example:

```
from nltk.tokenize import word_tokenize

text = "Natural Language Processing is amazing!"
tokens = word_tokenize(text)
print(tokens)
# Output:
['Natural', 'Language', 'Processing', 'is', 'amazing', '!']
```

Sentence Tokenization Example

- Sentence tokenization splits a text into individual sentences.
- Useful for tasks like summarization or sentiment analysis.

Python Example:

```
from nltk.tokenize import sent_tokenize

text = "Tokenization is fun. NLTK makes it easy!"
sentences = sent_tokenize(text)
print(sentences)
# Output: ['Tokenization is fun.', 'NLTK makes it easy!']
```

What is Part-of-Speech (POS) Tagging?

- **Definition:** POS tagging is the process of assigning grammatical categories (e.g., noun, verb, adjective) to each word in a text.
- POS tags provide insights into the structure and meaning of a sentence.
- Examples of common POS tags:
  - `NN` – Noun (e.g., "dog", "apple").
  - `VB` – Verb (e.g., "run", "is").
  - `JJ` – Adjective (e.g., "beautiful", "large").
  - `RB` – Adverb (e.g., "quickly", "very").

Why is POS tagging useful?

- Improves text processing tasks such as lemmatization, parsing, and information extraction.
- Provides the context necessary to interpret words accurately.

POS Tagging in NLTK

- NLTK provides the `pos_tag()` function to perform POS tagging.
- The function assigns a POS tag to each token in a sentence.

Python Example:

```
from nltk import pos_tag, word_tokenize

text = "The quick brown fox jumps over the lazy dog."
tokens = word_tokenize(text)
pos_tags = pos_tag(tokens)

print(pos_tags)
# Output:
# [('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'),
#  ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'),
#  ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN')]
```

Understanding POS Tags

- POS tags are abbreviations for grammatical categories.
- Common tags and their meanings:

Tag	Meaning
NN	Noun, singular or mass (e.g., "dog", "apple")
NNS	Noun, plural (e.g., "dogs")
VB	Verb, base form (e.g., "run")
VBD	Verb, past tense (e.g., "ran")
JJ	Adjective (e.g., "quick")
RB	Adverb (e.g., "quickly")
DT	Determiner (e.g., "the", "a")

- Full tagset: [NLTK POS Tag List](https://www.nltk.org/book/ch05.html).

Understanding POS Tags

- POS tags are abbreviations for grammatical categories.
- Common tags and their meanings:

Tag	Meaning
`NN`	Noun, singular or mass (e.g., "dog", "apple")
`NNS`	Noun, plural (e.g., "dogs")
`VB`	Verb, base form (e.g., "run")
`VBD`	Verb, past tense (e.g., "ran")
`JJ`	Adjective (e.g., "quick")
`RB`	Adverb (e.g., "quickly")
`DT`	Determiner (e.g., "the", "a")

- Full tagset: [NLTK POS Tag List](https://www.nltk.org/book/ch05.html).

Practice Session 1

- Tokenization ans POS-tagging with `nltk`

`https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks/-/blob/main/PythonIntroNotebooks/Lecture_07.ipynb`

What is Lemmatization?

- **Definition:** Lemmatization is the process of reducing a word to its dictionary form, or **lemma**, by considering its context and part of speech.
- Unlike stemming, lemmatization:
  - Produces valid words.
  - Considers the meaning of the word.
- Example:
  - "running" → "run".
  - "geese" → "goose".
  - "better" → "good".

Why Use Lemmatization?

- Preserves the meaning of words by reducing them to their canonical forms.
- Avoids ambiguity that might arise from stemming, which may produce non-words.
- Useful in:
  - Sentiment analysis.
  - Information retrieval.
  - Text classification.
- Examples:
  - "studies" → "study".
  - "better" → "good".

Lemmatization in NLTK

- NLTK uses `WordNetLemmatizer` for lemmatization.
- Lemmatization considers the part of speech (POS) to find the correct lemma.
- Example: Using `WordNetLemmatizer` in Python.

Python Example:

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

words = ["running", "geese", "better", "studies"]
lemmas = [lemmatizer.lemmatize(word) for word in words]

print(lemmas)
# Output: ['running', 'goose', 'better', 'study']
```

What is Stemming?

- **Stemming** is the process of reducing a word to its base or root form by removing prefixes and suffixes.
- Stemming may not always result in a valid word.
- Example: "running" → "run", "better" → "better".
- Common algorithms used for stemming:
  - Porter Stemmer.
  - Lancaster Stemmer.

Why Use Stemming?

- Helps to standardize words and reduce inflectional forms.
- Useful for simplifying text by mapping different forms of a word to a single token.
- Example:
  - "running", "runs", "runner" → "run".
- Stemming is often used in information retrieval and search engines.

Stemming in NLTK

- NLTK provides several stemmers, including:
  - `PorterStemmer`.
  - `LancasterStemmer`.
- Example: Using the Porter Stemmer in NLTK.

Python Example:

Code

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
words = ["running", "runs", "ran", "easily"]
stems = [stemmer.stem(word) for word in words]
print(stems)  # Output: ['run', 'run', 'ran', 'easili']
```

Sentiment Analysis

- Sentiment analysis is the process of identifying and classifying the emotional tone of a text.
- Common categories include:
  - **Positive**, **Negative**, **Neutral**
- Applications:
  - Analyzing customer reviews.
  - Monitoring social media sentiment.
  - Understanding public opinion.
- Example:
  - Text: *"This product is amazing!"*
  - Sentiment: **Positive**

Sentiment Analysis Pipeline

- Typical steps in a sentiment analysis pipeline:
  1. **Data Collection:** Gather text data (e.g., tweets, reviews).
  2. **Preprocessing:** Clean and tokenize the text.
  3. **Feature Extraction:** Convert text into numerical features (e.g., word counts, embeddings).
  4. **Classification:** Use a machine learning model to classify sentiment.
- Tools: NLTK, scikit-learn, pre-trained models (e.g., VADER, BERT).

Sentiment Analysis with CountVectorizer

- Converts text into a bag-of-words representation.
- Each text is represented as a vector where:
  - Columns correspond to unique words in the dataset.
  - Values are the frequency of each word in the text.
- Example:
  - Texts: `"I love cats"`, `"I love dogs"`

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

  - Columns: [`I`, `love`, `cats`, `dogs`]
- Converting text into numerical features using enables the application of standard ML algorithms.

Python Example:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

# Sample dataset
texts = ["I love this product!", "This is terrible.",
"It’s okay."]
labels = [1, 0, 1]  # 1: Positive, 0: Negative

# Transform texts to bag-of-words
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)

# Train classifier
classifier = LogisticRegression()
classifier.fit(X, labels)
```

Practice Session 2

Practice Session 2

- Lemmatization
- Sentiment analysis

`https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks/-/blob/main/PythonIntroNotebooks/Lecture_07.ipynb`

Summary of Topics Covered

- **Introduction to NLP and NLTK:**
  - Overview of NLP tasks and common techniques.
  - Introduction to NLTK for text processing.
- **Text Processing:**
  - Loading and cleaning text data.
  - Tokenization and basic text preprocessing.
- **Lemmatization:**
  - Using NLTK's lemmatizer to reduce words to their base forms.
- **Sentiment Analysis:**
  - Using CountVectorizer to convert text to features.
  - Training a classifier (e.g., Logistic Regression) for sentiment prediction.
- **NLP is a diverse field with many applications**

Summary of Topics Covered

- **Introduction to NLP and NLTK:**
  - Overview of NLP tasks and common techniques.
  - Introduction to NLTK for text processing.
- **Text Processing:**
  - Loading and cleaning text data.
  - Tokenization and basic text preprocessing.
- **Lemmatization:**
  - Using NLTK's lemmatizer to reduce words to their base forms.
- **Sentiment Analysis:**
  - Using CountVectorizer to convert text to features.
  - Training a classifier (e.g., Logistic Regression) for sentiment prediction.
- **NLP is a diverse field with many applications**

Exercise Session

`https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks/-/blob/main/PythonIntroNotebooks/Exercise_L07.ipynb`

Self-Study Questions: NLP with `nltk`

1. What is the difference between tokenization and lemmatization?
2. How can text data be cleaned and preprocessed for NLP tasks?
3. How does CountVectorizer represent text data numerically?
4. How would you modify the sentiment analysis example to use a different classifier (e.g., SVM)?
5. How do you use NLTK to lemmatize a word? What is the difference between stemming and lemmatization in NLTK?
6. What are some common challenges in working with text data for NLP tasks?
7. What is the purpose of Part-of-Speech tagging in NLP?