Comprehensive Introduction to Natural Language Processing (NLP)

A Clear and Practical Reference Guide

Prepared for Marouan

December 2, 2025

Contents

Chapter 1

Introduction

Natural Language Processing (NLP) is the field of artificial intelligence that allows computers to read, understand, and generate human language. This report explains all core NLP techniques using clear English and simple examples. It includes practical Python code and essential explanations.

The goal is to build strong foundations for future NLP work, from preprocessing to topic modeling to sentiment analysis.

Chapter 2

Text Preprocessing

Preprocessing prepares raw text so that models can use it.

2.1 Lowercasing

Lowercasing removes the difference between “Good” and “good”.

```
data['text'] = data['text'].str.lower()
```

2.2 Removing Punctuation

Punctuation rarely helps machine learning models.

```
import re
data['clean'] = data['text'].apply(lambda x:
    re.sub(r"([^\w\s])", "", x))
```

2.3 Stopword Removal

Stopwords like “the”, “and”, “is” are removed because they do not carry useful meaning.

```
from nltk.corpus import stopwords
en_stopwords = stopwords.words('english')

data['clean'] = data['clean'].apply(
    lambda x: '␣'.join([w for w in x.split()
                        if w not in en_stopwords])
)
```

2.4 Tokenization

Tokenization splits text into words.

```
from nltk.tokenize import word_tokenize
tokens = word_tokenize("Hello␣world!")
```

2.5 Stemming

Stemming reduces words to their root form by chopping.

Examples: “running → run“ “studies → studi“

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
stemmed = [ps.stem(w) for w in tokens]
```

2.6 Lemmatization

Lemmatization returns meaningful root words.

Examples: “running → run“ “better → good“

```
from nltk.stem import WordNetLemmatizer
lemm = WordNetLemmatizer()
lemm_words = [lemm.lemmatize(w) for w in tokens]
```

Chapter 3

Bag-of-Words (BoW)

BoW converts text into numbers by counting word frequency.

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer()
bow = cv.fit_transform(data)
```

BoW creates a matrix where each column is a word and each row is a document.

Chapter 4

TF-IDF

TF-IDF measures how important a word is in a document relative to the whole dataset.

Important words: – appear often in one document – appear rarely in others

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
tfidf_matrix = tfidf.fit_transform(data)
```

Chapter 5

N-grams

Instead of single words, we can use sequences of words:

Unigram: 1 word Bigram: 2 words ("great hotel") Trigram: 3 words ("great hotel staff")

```
import nltk
bigrams = list(nltk.ngrams(tokens, 2))
```

Chapter 6

Part-of-Speech (POS) Tagging

POS tagging identifies the role of each word: noun, verb, adjective, etc.

```
import spacy
nlp = spacy.load("en_core_web_sm")

doc = nlp("Emma␣is␣reading␣a␣book.")
for token in doc:
    print(token.text, token.pos_)
```

Chapter 7

Topic Modeling: LDA

Latent Dirichlet Allocation (LDA) discovers hidden topics in text.

```
import gensim
from gensim import corpora

dictionary = corpora.Dictionary(articles)
corpus = [dictionary.doc2bow(text) for text in articles]

lda_model = gensim.models.LdaModel(
    corpus=corpus, id2word=dictionary, num_topics=2)
```

LDA outputs topics as groups of important words.

Chapter 8

Topic Modeling: LSA

Latent Semantic Analysis (LSA) uses matrix decomposition to extract hidden relationships.

```
from gensim.models import LsiModel
lsa_model = LsiModel(corpus, num_topics=2, id2word=dictionary)
```

Chapter 9

Sentiment Analysis

9.1 TextBlob

TextBlob gives a polarity score between 1 and +1.

```
from textblob import TextBlob
TextBlob("I␣love␣this").sentiment
```

9.2 VADER

VADER is excellent for social media or short comments.

```
from vaderSentiment.vaderSentiment import
   SentimentIntensityAnalyzer

analyzer = SentimentIntensityAnalyzer()
analyzer.polarity_scores("The␣movie␣was␣great!")
```

9.3 Transformers

Transformers understand context much better than classical methods.

```
from transformers import pipeline
sentiment = pipeline("sentiment-analysis")

sentiment("I␣love␣this␣movie!")
```

Chapter 10

Machine Learning Classifiers

10.1 Logistic Regression

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
```

10.2 Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB().fit(X_train, y_train)
```

10.3 Support Vector Machines (SVM)

```
from sklearn.linear_model import SGDClassifier

svm = SGDClassifier().fit(X_train, y_train)
```

Chapter 11

Topic Modeling Evaluation

Coherence scores help choose the optimal number of topics.

```
from gensim.models.coherencemodel import CoherenceModel

coh = CoherenceModel(model=lda_model,
                     texts=articles,
                     dictionary=dictionary,
                     coherence='c_v')
coh.get_coherence()
```

Chapter 12

Complete NLP Pipeline Example

12.1 Steps

1. Load data 2. Clean text 3. Remove stopwords 4. Tokenize 5. Stem or lemmatize 6. Convert to vectors (BoW / TF-IDF) 7. Train model 8. Evaluate 9. Predict

12.2 Python Mini-Pipeline

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer

# vectorize text
tfidf = TfidfVectorizer()
X = tfidf.fit_transform(data['text'])
y = data['label']

# split
X_train, X_test, y_train, y_test = train_test_split(X, y,
   test_size=0.2)

# model
model = LogisticRegression()
model.fit(X_train, y_train)

# evaluation
pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, pred))
```

Chapter 13

Conclusion

This document summarizes the essential concepts of NLP:

• Preprocessing • BoW and TF-IDF • N-grams • POS tagging • Topic modeling (LDA, LSA) • Sentiment analysis • Classical ML classifiers • Transformer models • Evaluation methods

These topics form the foundation of modern NLP and are necessary for advanced tasks such as text classification, chatbots, translation, and large language model fine-tuning.