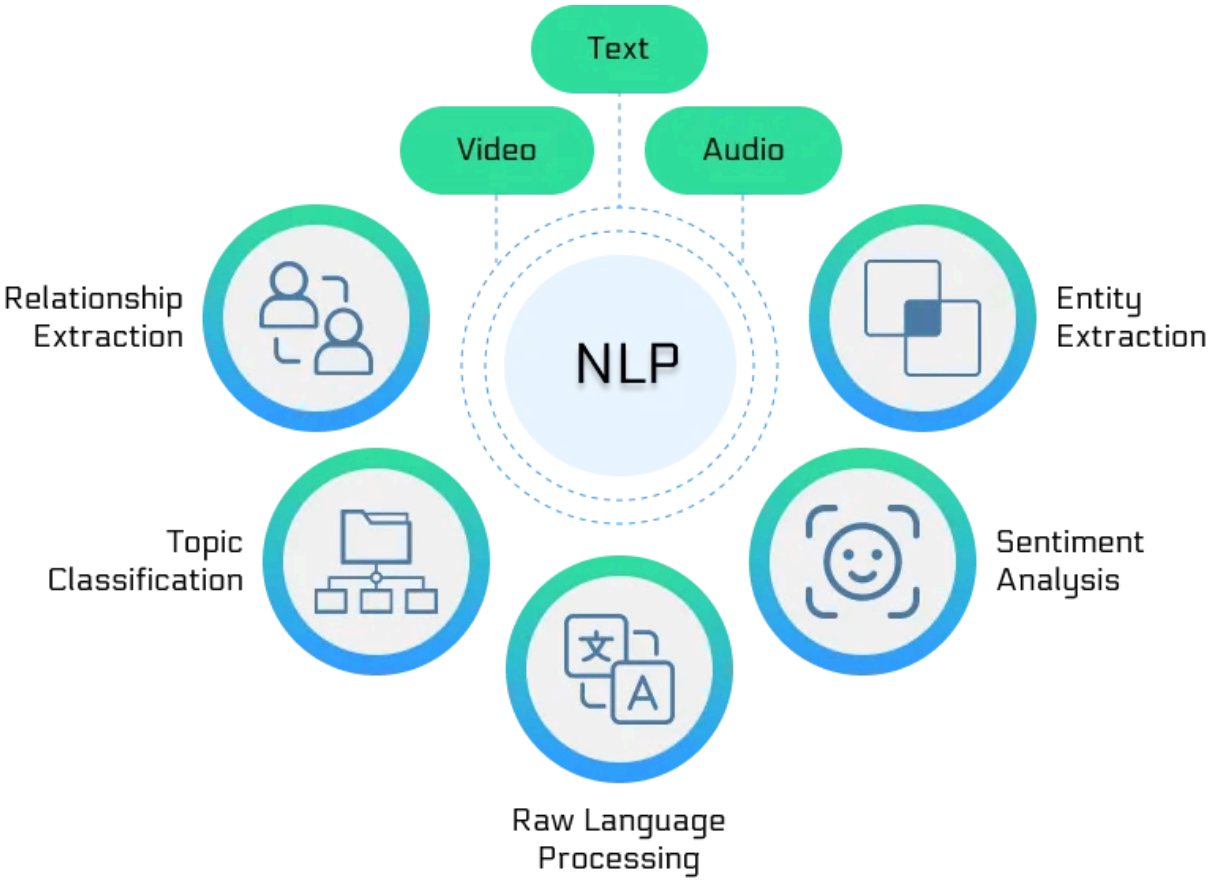
Natural Language Processing (NLP) Cheat Sheet with Python

5 min read · May 6, 2025

Statfusionai Follow

Listen Share



Key Libraries

- **NLTK:** Traditional NLP toolkit with extensive linguistic resources
- **spaCy:** Industrial-strength NLP with efficient processing pipelines

- **Transformers:** Hugging Face library for state-of-the-art models (BERT, GPT, etc.)
- **Gensim:** Topic modeling and document similarity
- **TextBlob:** Simple, intuitive interface for common NLP tasks
- **scikit-learn:** Machine learning algorithms for text classification

Text Preprocessing

Text Cleaning (NLTK)

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

```
# Download resources
nltk.download('punkt')
nltk.download('stopwords')
def clean_text(text):
    # Lowercase
    text = text.lower()

    # Remove special characters
    text = re.sub(r'[^\w\s]', '', text)

    # Tokenize
    tokens = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [w for w in tokens if w not in stop_words]

    return filtered_tokens
```

Tokenization

NLTK:

```
from nltk.tokenize import word_tokenize, sent_tokenize
# Sentence tokenization
sentences = sent_tokenize("Hello there. How are you doing today?")
# ['Hello there.', 'How are you doing today?']
# Word tokenization
words = word_tokenize("Natural language processing is fascinating!")
# ['Natural', 'language', 'processing', 'is', 'fascinating', '!']
```

spaCy:

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Natural language processing is fascinating!")
# Word tokens
tokens = [token.text for token in doc]
# ['Natural', 'language', 'processing', 'is', 'fascinating', '!']
# Sentence tokens
sentences = [sent.text for sent in doc.sents
```

Stemming & Lemmatization

NLTK Stemming:

```
from nltk.stem import PorterStemmer, LancasterStemmer, SnowballStemmer
porter = PorterStemmer()
lancaster = LancasterStemmer()
snowball = SnowballStemmer('english')
word = "running"
print(porter.stem(word))     # run
print(lancaster.stem(word))  # run
print(snowball.stem(word))   # run
```

NLTK Lemmatization:

```
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("better", pos='a'))  # good
print(lemmatizer.lemmatize("running", pos='v')) # run
print(lemmatizer.lemmatize("mice"))
```

spaCy Lemmatization:

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("I am running in the park")
# Get lemmas
lemmas = [token.lemma_ for token in doc]
# ['I', 'be', 'run', 'in', 'the', 'park']
```

Feature Extraction

Bag of Words

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = [
    "Natural language processing is fascinating.",
    "I love learning about NLP and machine learning."
]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
# Get feature names
print(vectorizer.get_feature_names_out())
# ['about', 'and', 'fascinating', 'is', 'language', ...]
# Get document vectors
print(X.toarray())
# [[0 0 1 1 1 ...], [1 1 0 0 0 ...]
```

TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = [
    "Natural language processing is fascinating.",
    "I love learning about NLP and machine learning."
]
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)
# Get feature names
print(vectorizer.get_feature_names_out())
# Get document vectors with tf-idf weights
print(X.toarray())
```

Word Embeddings

Word2Vec with Gensim:

```
from gensim.models import Word2Vec
# List of tokenized sentences
sentences = [["natural", "language", "processing"],
             ["machine", "learning", "algorithms"]]
# Train model
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, sg
# Get vector for word
vector = model.wv['natural']
# Find similar words
similar = model.wv.most_similar('natural', topn=5)
```

Using pre-trained embeddings:

```
import gensim.downloader as api
# Load pre-trained model
word_vectors = api.load("glove-wiki-gigaword-100")
# Get vector for word
vector = word_vectors['computer']
# Find similar words
similar = word_vectors.most_similar('computer', topn=5)
```

Text Classification

Simple Classifier with scikit-learn

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
# Sample data
X_train = ["I love this product", "This is terrible", "Great experienc
y_train = ["positive", "negative", "positive"]
# Create pipeline
text_clf = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', MultinomialNB())
])
# Train
text_clf.fit(X_train, y_train)
# Predict
text_clf.predict(["This was awesome"])  # ['positive']
```

Using Transformers

```
from transformers import pipeline
# Load sentiment analysis pipeline
classifier = pipeline('sentiment-analysis')
# Classify text
result = classifier("I've been waiting for this movie for years!")
# [{'label': 'POSITIVE', 'score': 0.9998}]
```

Named Entity Recognition (NER)

spaCy NER:

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
# Apple ORG
# U.K. GPE
# $1 billion MONEY
```

NLTK NER:

```
import nltk
from nltk import word_tokenize, pos_tag, ne_chunk
nltk.download('maxent_ne_chunker')
nltk.download('words')
sentence = "Mark Zuckerberg is the CEO of Facebook."
tokens = word_tokenize(sentence)
tagged = pos_tag(tokens)
entities = ne_chunk(tagged)
print(entities)
```

Topic Modeling

Latent Dirichlet Allocation (LDA) with Gensim:

```
from gensim.corpora import Dictionary
from gensim.models import LdaModel
import gensim.corpora as corpora
# Sample documents
docs = [
    "Machine learning is a subset of artificial intelligence",
    "NLP is used for text analysis and understanding",
    "Deep learning networks have multiple hidden layers"
]
# Tokenize
tokenized_docs = [doc.lower().split() for doc in docs]
# Create dictionary
dictionary = Dictionary(tokenized_docs)
# Create corpus
corpus = [dictionary.doc2bow(doc) for doc in tokenized_docs]
# Train LDA model
lda_model = LdaModel(
    corpus=corpus,
    id2word=dictionary,
    num_topics=2,
```

```
    passes=10
)
# Print topics
topics = lda_model.print_topics()
for topic in topics:
    print(topic)
```

Sentiment Analysis

TextBlob:

```
from textblob import TextBlob
text = "The movie was absolutely amazing! I loved it."
blob = TextBlob(text)
# Polarity ranges from -1 (negative) to 1 (positive)
print(blob.sentiment.polarity)  # 0.8
# Subjectivity ranges from 0 (objective) to 1 (subjective)
print(blob.sentiment.subjectivity)  # 0.75
```

VADER Sentiment (NLTK):

```
from nltk.sentiment import SentimentIntensityAnalyzer
import nltk
nltk.download('vader_lexicon')
sia = SentimentIntensityAnalyzer()
text = "The movie was absolutely amazing! I loved it."
scores = sia.polarity_scores(text)
print(scores)
# {'neg': 0.0, 'neu': 0.295, 'pos': 0.705, 'compound': 0.8316}
```

Text Summarization

Extractive Summarization with sumy:

```
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lex_rank import LexRankSummarizer
```

```
text = """Natural language processing (NLP) is a subfield of linguisti
# Initialize the parser
parser = PlaintextParser.from_string(text, Tokenizer("english"))
# Initialize the summarizer
summarizer = LexRankSummarizer()
# Summarize
summary = summarizer(parser.document, sentences_count=1)
# Print summary
for sentence in summary:
    print(sentence)
```

Language Translation

Using Transformers:

```
from transformers import pipeline
# Initialize translator
translator = pipeline("translation_en_to_fr")
# Translate
translation = translator("Hello, how are you?")
print(translation[0]['translation_text'])  # 'Bonjour, comment allez-v
```

Key Tips for NLP Projects

1. **Start simple:** Try basic models before complex ones
2. **Clean your data thoroughly:** Good preprocessing is crucial
3. **Consider context:** Many NLP problems need contextual understanding
4. **Evaluate properly:** Use appropriate metrics (accuracy, F1, BLEU, etc.)
5. **Balance efficiency and accuracy:** Choose libraries that fit your needs
6. **Leverage pre-trained models:** They often outperform models trained from scratch
7. **Handle imbalanced data:** Use techniques like oversampling or adjusted class weights