Natural Language Processing with Python

A Comprehensive Cheat Sheet for NLP Tasks and Techniques

🔤 **NLTK**

🚀 **spaCy**

🤗 **Transformers**

📊 **Gensim**

🧠 **sklearn**

Text Preprocessing

▶ Cleaning & Tokenization NLTK

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

nltk.download('punkt')
nltk.download('stopwords')

def clean_text(text):
  text = text.lower()
  text = re.sub(r'[^\w\s]', '', text)
  tokens = word_tokenize(text)
  stop_words = set(stopwords.words('english'))
  return [w for w in tokens if w not in stop_words]
```

▶ Stemming vs Lemmatization

```
# Stemming with NLTK
from nltk.stem import PorterStemmer
porter = PorterStemmer()
porter.stem("running") # "run"

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

Preprocessing Tips

- Always lowercase text for consistency
- Remove stopwords for topic modeling, but keep them for sentiment analysis
- Use lemmatization over stemming when meaning preservation is important
- Consider domain-specific preprocessing (e.g., hashtags for social media)

Feature Extraction

▶ Bag of Words `sklearn`

```
from sklearn.feature_extraction.text import CountVectorizer

corpus = [
  "Natural language processing.",
  "I love learning about NLP."
]

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
# Get feature names & document vectors
vectorizer.get_feature_names_out()
X.toarray()
```

▶ TF-IDF `sklearn`

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)
```

▶ Word Embeddings `Gensim`

```
from gensim.models import Word2Vec

sentences = [["natural", "language"], ["machine", "learning"]]

model = Word2Vec(sentences, vector_size=100, window=5, min_count=1)

# Get vector & similar words
vector = model.wv['natural']
similar = model.wv.most_similar('natural', topn=5)
```

When to Use Each Feature Type

- **BoW/TF-IDF:** Text classification, document clustering
- **Word Embeddings:** Semantic tasks, text similarity, transfer learning

- **Contextual Embeddings:** Advanced tasks requiring context understanding

Text Classification

▶ Basic Pipeline sklearn

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline

X_train = ["I love this product", "This is terrible"]
y_train = ["positive", "negative"]

text_clf = Pipeline([
  ('tfidf', TfidfVectorizer()),
  ('clf', MultinomialNB())
])

text_clf.fit(X_train, y_train)
text_clf.predict(["This was awesome"]) # ['positive']
```

▶ Using Transformers Transformers

```
from transformers import pipeline

classifier = pipeline('sentiment-analysis')

result = classifier("I've been waiting for this movie!")
# [{'label': 'POSITIVE', 'score': 0.9998}]
```

Classification Task	Recommended Approach
Sentiment Analysis	VADER (rule-based) or fine-tuned BERT
Topic Classification	TF-IDF + SVM or DistilBERT
Intent Recognition	Fine-tuned RoBERTa
Spam Detection	TF-IDF + Naive Bayes

Named Entity Recognition

▶ spaCy NER spaCy

```python
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is 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
```

▶ Transformers NER Transformers

```python
from transformers import pipeline

ner = pipeline("ner")

text = "My name is Sarah and I work at Google in London"
ner_results = ner(text)
# [{'entity': 'I-PER', 'score': 0.99, 'word': 'Sarah'}, ...]
```

Common Entity Types

- **PER/PERSON:** People names
- **ORG:** Organizations, companies
- **LOC/GPE:** Locations, geopolitical entities
- **DATE/TIME:** Temporal expressions
- **MONEY:** Monetary values

Sentiment Analysis

▶ TextBlob TextBlob

```python
from textblob import TextBlob

text = "The movie was absolutely amazing!"
blob = TextBlob(text)

# Polarity: -1 (negative) to 1 (positive)
print(blob.sentiment.polarity) # 0.8

# Subjectivity: 0 (objective) to 1 (subjective)
print(blob.sentiment.subjectivity) # 0.75
```

▶ VADER Sentiment NLTK

```python
from nltk.sentiment import SentimentIntensityAnalyzer
import nltk

nltk.download('vader_lexicon')
sia = SentimentIntensityAnalyzer()

text = "The movie was absolutely amazing!"
scores = sia.polarity_scores(text)

print(scores)
# {'neg': 0.0, 'neu': 0.295, 'pos': 0.705, 'compound': 0.8316}
```

Sentiment Analysis Tips

- Rule-based approaches work well for straightforward text
- Consider using domain-specific models for specialized content
- ML/DL approaches handle context, sarcasm, and negation better
- Use BERT variants for state-of-the-art performance

Topic Modeling

▶ LDA with Gensim `Gensim`

```python
from gensim.corpora import Dictionary
from gensim.models import LdaModel

docs = [
  "Machine learning is a subset of AI",
  "NLP is used for text analysis"
]

# Tokenize
tokenized_docs = [doc.lower().split() for doc in docs]

# Create dictionary & corpus
dictionary = Dictionary(tokenized_docs)
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)
```

Topic Modeling Approaches

- **LDA:** Classic probabilistic approach
- **NMF:** Non-negative Matrix Factorization
- **BERTopic:** Leverages BERT embeddings
- **Top2Vec:** Document embeddings + clustering

Advanced Techniques

▶ Text Summarization Transformers

```
from transformers import pipeline

summarizer = pipeline("summarization")

long_text = """NLP is a field of AI that focuses on..."""

summary = summarizer(long_text, max_length=100, min_length=30)
print(summary[0]['summary_text'])
```

▶ Translation Transformers

```
from transformers import pipeline

translator = pipeline("translation_en_to_fr")

translation = translator("Hello, how are you?")
print(translation[0]['translation_text'])
```

▶ Question Answering Transformers

```
from transformers import pipeline

qa = pipeline("question-answering")

context = "Python is a programming language created by..."
question = "Who created Python?"

result = qa(question=question, context=context)
print(result['answer'])
```

Task	Beginner Approach	Advanced Approach
Summarization	Extractive (TextRank)	Abstractive (T5, BART)
Translation	Pre-trained pipeline	Custom Seq2Seq models

Q&A	Rule-based systems	Fine-tuned BERT/T5
Text Generation	Markov Chains	GPT models

NLP Project Evaluation & Tips

Evaluation Metrics by Task

- **Classification:** Accuracy, F1-score, Precision, Recall
- **NER:** F1-score, Precision, Recall (by entity type)
- **Summarization:** ROUGE-N, ROUGE-L, BLEU
- **Translation:** BLEU, METEOR, TER
- **Generation:** Perplexity, human evaluation

Best Practices for NLP Projects

- Start simple: Try basic models before complex ones
- Clean your data thoroughly: Good preprocessing is crucial
- Consider context: Many NLP problems need contextual understanding
- Leverage pre-trained models: Often outperform models trained from scratch
- Handle class imbalance: Use oversampling or adjusted weights
- Use cross-validation: Especially for small datasets
- Evaluate properly: Choose appropriate metrics for your task