Session 1: Complete NLP Introduction & Text Pre-processing Guide

📚 Table of Contents

🎯 Introduction to Natural Language Processing

What is NLP?

Natural Language Processing (NLP) is a branch of artificial intelligence that focuses on the interaction between computers and humans through natural language. The ultimate objective is to read, decipher, understand, and make sense of human language in a manner that is valuable.

Key Components of NLP:

- **Natural Language Understanding (NLU)**: Machine's ability to understand human language
- **Natural Language Generation (NLG)**: Machine's ability to generate human-like text
- **Natural Language Interaction**: Combination of both understanding and generation

Why Do We Need NLP?

1. **Data Volume**: 80% of world's data is unstructured text
2. **Human Communication**: We naturally communicate through language
3. **Automation**: Need to automate text-based tasks
4. **Insights**: Extract meaningful information from text data
5. **Accessibility**: Make technology more accessible through natural interaction

Basic NLP Pipeline:

```
Raw Text → Preprocessing → Feature Extraction → Model Training → Prediction → Post-
processing
```

🌟 Applications of NLP

1. Real-World Applications

Search Engines

- Google's search algorithm uses NLP to understand queries
- Semantic search for better results

Virtual Assistants

- Siri, Alexa, Google Assistant
- Voice recognition and response generation

Social Media Monitoring

- Sentiment analysis of brand mentions
- Trend detection and analysis

Healthcare

- Medical record analysis
- Drug discovery through literature mining

Finance

- Fraud detection through text analysis
- Automated trading based on news sentiment

Customer Service

- Chatbots for customer support
- Automated ticket classification

2. Code Example: Simple NLP Application Detection

python

```python
import re
import nltk
from collections import Counter

def detect_application_type(text):
    """
    Simple function to detect what type of NLP application might be needed
    """
    text_lower = text.lower()

    # Define keywords for different applications
    sentiment_keywords = ['good', 'bad', 'love', 'hate', 'amazing', 'terrible', 'excellent', 'a
    search_keywords = ['find', 'search', 'look for', 'where is', 'what is', 'how to']
    translation_keywords = ['translate', 'spanish', 'french', 'german', 'chinese', 'japanese']

    # Check for sentiment analysis needs
    sentiment_score = sum(1 for word in sentiment_keywords if word in text_lower)

    # Check for search needs
    search_score = sum(1 for phrase in search_keywords if phrase in text_lower)

    # Check for translation needs
    translation_score = sum(1 for word in translation_keywords if word in text_lower)

    scores = {
        'sentiment_analysis': sentiment_score,
        'search': search_score,
        'translation': translation_score
    }

    if max(scores.values()) == 0:
        return 'general_nlp'

    return max(scores.keys(), key=scores.get)

# Examples
texts = [
    "I love this product, it's amazing!",
    "How to find the best restaurant nearby?",
    "Please translate this text to Spanish",
    "The weather forecast for tomorrow"
]

for text in texts:
    app_type = detect_application_type(text)
    print(f"Text: '{text}' -> Application: {app_type}")
```

🛠️ Text Pre-processing Theory

Why is Text Pre-processing Important?

1. **Noise Reduction**: Remove irrelevant information
2. **Standardization**: Create consistent format
3. **Computational Efficiency**: Reduce processing time
4. **Model Performance**: Improve accuracy of ML models
5. **Feature Engineering**: Create better input features

Common Challenges with Text Data:

1. Inconsistency Issues

- Different spellings: "color" vs "colour"
- Case variations: "Apple" vs "apple"
- Abbreviations: "Dr." vs "Doctor"

2. Noise and Irrelevant Information

- HTML tags in web-scraped text
- Special characters and punctuation
- Extra whitespaces

3. Language Complexities

- Synonyms: "big" and "large"
- Polysemy: "bank" (financial institution vs river bank)
- Context dependency

4. Encoding Issues

- Different character encodings (UTF-8, ASCII)
- Emoji and special unicode characters

Text Pre-processing Steps Overview:

1. **Text Cleaning**: Remove noise and irrelevant characters
2. **Tokenization**: Split text into individual words/tokens

3. **Normalization**: Convert to standard form
4. **Stop Words Removal**: Remove common words
5. **Stemming/Lemmatization**: Reduce words to root form
6. **Feature Engineering**: Create numerical representations

🔧 Complete Text Pre-processing Implementation

Step 1: Environment Setup

python

```python
# Required installations
# pip install nltk spacy beautifulsoup4 wordcloud matplotlib

import re
import string
import nltk
import spacy
import pandas as pd
import numpy as np
from collections import Counter
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import PorterStemmer, SnowballStemmer, WordNetLemmatizer
from nltk.chunk import ne_chunk
from nltk.tag import pos_tag
import matplotlib.pyplot as plt
from wordcloud import WordCloud

# Download required NLTK data
nltk_downloads = [
    'punkt', 'stopwords', 'wordnet', 'averaged_perceptron_tagger',
    'maxent_ne_chunker', 'words', 'vader_lexicon'
]

for item in nltk_downloads:
    try:
        nltk.download(item, quiet=True)
    except:
        print(f"Could not download {item}")

# Load spaCy model (install with: python -m spacy download en_core_web_sm)
try:
    nlp = spacy.load("en_core_web_sm")
except OSError:
    print("spaCy model not found. Install with: python -m spacy download en_core_web_sm")
    nlp = None
```

Step 2: Basic Text Cleaning Functions

python

```python
class TextCleaner:
    def __init__(self):
        self.stop_words = set(stopwords.words('english'))
        self.stemmer = PorterStemmer()
        self.lemmatizer = WordNetLemmatizer()

    def clean_text(self, text, remove_digits=True, remove_punct=True):
        """
        Basic text cleaning function
        """
        if not isinstance(text, str):
            return ""

        # Convert to lowercase
        text = text.lower()

        # Remove URLs
        text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

        # Remove email addresses
        text = re.sub(r'\S+@\S+', '', text)

        # Remove user mentions (Twitter style)
        text = re.sub(r'@\w+', '', text)

        # Remove hashtags but keep the text
        text = re.sub(r'#(\w+)', r'\1', text)

        # Remove HTML tags
        text = re.sub(r'<.*?>', '', text)

        # Remove digits if specified
        if remove_digits:
            text = re.sub(r'\d+', '', text)

        # Remove punctuation if specified
        if remove_punct:
            text = text.translate(str.maketrans('', '', string.punctuation))

        # Remove extra whitespace
        text = ' '.join(text.split())

        return text

    def advanced_clean(self, text):
        """
        Advanced cleaning with more specific patterns
        """
        # Handle contractions
        contractions = {
            "won't": "will not", "can't": "cannot", "n't": " not",
            "'re": " are", "'ve": " have", "'ll": " will",
            "'d": " would", "'m": " am", "'s": " is"
        }

        for contraction, expansion in contractions.items():
            text = text.replace(contraction, expansion)

        # Remove repeated characters (e.g., "sooooo" -> "so")
        text = re.sub(r'(.)\1{2,}', r'\1\1', text)

        # Remove standalone numbers and special characters
        text = re.sub(r'\b\d+\b', '', text)
        text = re.sub(r'[^\w\s]', '', text)

        # Remove extra whitespace
        text = ' '.join(text.split())

        return text

# Example usage
cleaner = TextCleaner()

sample_texts = [
    "Hello World!!! This is an AMAZING example text. Visit https://example.com @user #NLP",
    "I can't believe it's sooooo good! 😊 Check email: test@example.com",
    "HTML content: <div>Some text</div> with numbers: 12345",
    "Don't you think this won't work? I'm not sure..."
]

print("Basic Text Cleaning Examples:")
print("-" * 50)
for i, text in enumerate(sample_texts, 1):
    cleaned = cleaner.clean_text(text)
    advanced = cleaner.advanced_clean(cleaned)
    print(f"Original {i}: {text}")
    print(f"Cleaned {i}: {cleaned}")
    print(f"Advanced {i}: {advanced}")
    print()
```

Step 3: Tokenization Techniques

python

```python
class Tokenizer:
    def __init__(self):
        pass

    def word_tokenize_nltk(self, text):
        """NLTK word tokenization"""
        return word_tokenize(text)

    def sentence_tokenize_nltk(self, text):
        """NLTK sentence tokenization"""
        return sent_tokenize(text)

    def spacy_tokenize(self, text):
        """spaCy tokenization with linguistic features"""
        if nlp is None:
            return []

        doc = nlp(text)
        tokens_info = []

        for token in doc:
            tokens_info.append({
                'text': token.text,
                'lemma': token.lemma_,
                'pos': token.pos_,
                'tag': token.tag_,
                'is_alpha': token.is_alpha,
                'is_stop': token.is_stop,
                'is_punct': token.is_punct
            })

        return tokens_info

    def custom_tokenize(self, text, pattern=r'\b\w+\b'):
        """Custom regex-based tokenization"""
        return re.findall(pattern, text.lower())

    def advanced_tokenize(self, text):
        """Advanced tokenization preserving important elements"""
        # Preserve emails, URLs, mentions, hashtags
        special_tokens = []

        # Extract and replace special tokens
        patterns = {
            'email': r'\S+@\S+',
            'url': r'http\S+|www\S+',
            'mention': r'@\w+',
            'hashtag': r'#\w+'
        }

        processed_text = text
        for token_type, pattern in patterns.items():
            matches = re.findall(pattern, text)
            for i, match in enumerate(matches):
                placeholder = f"__{token_type.upper()}_{i}__"
                special_tokens.append((placeholder, match))
                processed_text = processed_text.replace(match, placeholder)

        # Regular tokenization
        tokens = word_tokenize(processed_text)

        # Restore special tokens
        for placeholder, original in special_tokens:
            if placeholder in tokens:
                idx = tokens.index(placeholder)
                tokens[idx] = original

        return tokens

# Example usage
tokenizer = Tokenizer()

text = "Hello! Visit https://example.com or email test@example.com. Follow @user and check #NLP

print("Tokenization Examples:")
print("-" * 50)
print(f"Original text: {text}")
print()

# NLTK word tokenization
nltk_tokens = tokenizer.word_tokenize_nltk(text)
print(f"NLTK tokens: {nltk_tokens}")

# NLTK sentence tokenization
sentences = tokenizer.sentence_tokenize_nltk(text)
print(f"Sentences: {sentences}")

# spaCy tokenization (if available)
if nlp:
    spacy_tokens = tokenizer.spacy_tokenize(text)
    print(f"spaCy tokens (first 3): {spacy_tokens[:3]}")

# Custom tokenization
```

```python
custom_tokens = tokenizer.custom_tokenize(text)
print(f"Custom tokens: {custom_tokens}")

# Advanced tokenization
advanced_tokens = tokenizer.advanced_tokenize(text)
print(f"Advanced tokens: {advanced_tokens}")
```

Step 4: Stop Words Handling

python

```python
class StopWordsHandler:
    def __init__(self, language='english'):
        self.nltk_stopwords = set(stopwords.words(language))
        self.custom_stopwords = set()

        # Common additional stop words for specific domains
        self.domain_stopwords = {
            'social_media': {'rt', 'via', 'dm', 'ff', 'tbt', 'imo', 'imho'},
            'web': {'click', 'here', 'link', 'page', 'site', 'web', 'www'},
            'academic': {'paper', 'study', 'research', 'analysis', 'results'},
            'business': {'company', 'business', 'corp', 'inc', 'ltd'}
        }

    def get_stopwords(self, include_domains=None):
        """Get combined stop words"""
        all_stopwords = self.nltk_stopwords.copy()
        all_stopwords.update(self.custom_stopwords)

        if include_domains:
            for domain in include_domains:
                if domain in self.domain_stopwords:
                    all_stopwords.update(self.domain_stopwords[domain])

        return all_stopwords

    def add_custom_stopwords(self, words):
        """Add custom stop words"""
        if isinstance(words, str):
            words = [words]
        self.custom_stopwords.update(words)

    def remove_stopwords(self, tokens, include_domains=None):
        """Remove stop words from token list"""
        stopwords_set = self.get_stopwords(include_domains)
        return [token for token in tokens if token.lower() not in stopwords_set]

    def analyze_stopwords(self, tokens):
        """Analyze stop word distribution"""
        stopwords_set = self.get_stopwords()

        stop_count = sum(1 for token in tokens if token.lower() in stopwords_set)
        content_count = len(tokens) - stop_count

        analysis = {
            'total_tokens': len(tokens),
            'stop_words': stop_count,
            'content_words': content_count,
            'stop_word_ratio': stop_count / len(tokens) if tokens else 0
        }

        return analysis

# Example usage
stop_handler = StopWordsHandler()

# Add custom stop words
stop_handler.add_custom_stopwords(['said', 'says', 'think', 'know'])

text = "I think this is a great example of natural language processing. I said it before and I'
tokens = word_tokenize(text.lower())

print("Stop Words Analysis:")
print("-" * 50)
print(f"Original tokens: {tokens}")

# Basic stop word removal
filtered_basic = stop_handler.remove_stopwords(tokens)
print(f"After removing stop words: {filtered_basic}")

# With domain-specific stop words
filtered_social = stop_handler.remove_stopwords(tokens, include_domains=['social_media'])
print(f"With social media stop words: {filtered_social}")

# Analysis
analysis = stop_handler.analyze_stopwords(tokens)
print(f"Stop word analysis: {analysis}")

# Show which words were removed
removed_words = [token for token in tokens if token not in filtered_basic]
print(f"Removed stop words: {removed_words}")
```

Step 5: Stemming and Lemmatization

```
python
```

```python
class WordNormalizer:
    def __init__(self):
        # Different stemmers
        self.porter_stemmer = PorterStemmer()
        self.snowball_stemmer = SnowballStemmer('english')
        self.lemmatizer = WordNetLemmatizer()

    def stem_words(self, tokens, stemmer_type='porter'):
        """Apply stemming to tokens"""
        if stemmer_type == 'porter':
            return [self.porter_stemmer.stem(token) for token in tokens]
        elif stemmer_type == 'snowball':
            return [self.snowball_stemmer.stem(token) for token in tokens]
        else:
            raise ValueError("Unknown stemmer type")

    def lemmatize_words(self, tokens, pos_tags=None):
        """Apply lemmatization to tokens"""
        if pos_tags is None:
            # Simple lemmatization without POS tags
            return [self.lemmatizer.lemmatize(token) for token in tokens]
        else:
            # Lemmatization with POS tags for better accuracy
            lemmas = []
            for token, pos in zip(tokens, pos_tags):
                # Convert POS tag to WordNet format
                wordnet_pos = self._get_wordnet_pos(pos)
                lemmas.append(self.lemmatizer.lemmatize(token, wordnet_pos))
            return lemmas

    def _get_wordnet_pos(self, treebank_tag):
        """Convert POS tag to WordNet format"""
        if treebank_tag.startswith('J'):
            return 'a'  # adjective
        elif treebank_tag.startswith('V'):
            return 'v'  # verb
        elif treebank_tag.startswith('N'):
            return 'n'  # noun
        elif treebank_tag.startswith('R'):
            return 'r'  # adverb
        else:
            return 'n'  # default to noun

    def compare_normalization(self, tokens):
        """Compare different normalization techniques"""
        # Get POS tags for better lemmatization
        pos_tags = pos_tag(tokens)
        pos_only = [pos for token, pos in pos_tags]

        results = {
            'original': tokens,
            'porter_stem': self.stem_words(tokens, 'porter'),
            'snowball_stem': self.stem_words(tokens, 'snowball'),
            'lemma_simple': self.lemmatize_words(tokens),
            'lemma_pos': self.lemmatize_words(tokens, pos_only)
        }

        return results

    def analyze_reduction(self, original_tokens, normalized_tokens):
        """Analyze vocabulary reduction after normalization"""
        original_vocab = set(original_tokens)
        normalized_vocab = set(normalized_tokens)

        reduction_stats = {
            'original_vocab_size': len(original_vocab),
            'normalized_vocab_size': len(normalized_vocab),
            'reduction_count': len(original_vocab) - len(normalized_vocab),
            'reduction_percentage': (len(original_vocab) - len(normalized_vocab)) / len(origina
        }

        return reduction_stats

# Example usage
normalizer = WordNormalizer()

# Test with various word forms
test_words = [
    'running', 'ran', 'runs', 'runner',
    'better', 'good', 'best',
    'flying', 'flies', 'flew',
    'children', 'child',
    'went', 'going', 'gone'
]

print("Word Normalization Comparison:")
print("-" * 70)

# Compare normalization techniques
comparison = normalizer.compare_normalization(test_words)

# Create a comparison table
df = pd.DataFrame(comparison)
```

```
print(df.to_string(index=False))

print("\nDetailed Analysis:")
print("-" * 30)

# Analyze reduction for each technique
for method in ['porter_stem', 'snowball_stem', 'lemma_simple', 'lemma_pos']:
    stats = normalizer.analyze_reduction(test_words, comparison[method])
    print(f"{method.replace('_', ' ').title()}:")
    print(f"  Vocabulary reduction: {stats['reduction_count']} words ({stats['reduction_percent

# Show specific examples
print("\nSpecific Examples:")
print("-" * 20)
examples = [
    ('running', 'runs', 'ran'),
    ('better', 'good', 'best'),
    ('children', 'child'),
    ('went', 'going', 'gone')
]

for example_group in examples:
    print(f"\nWord group: {example_group}")
    group_comparison = normalizer.compare_normalization(list(example_group))
    for method, results in group_comparison.items():
        if method != 'original':
            print(f"  {method}: {results}")
```

Step 6: Complete Text Preprocessing Pipeline

python

```python
class TextPreprocessor:
    def __init__(self,
                 language='english',
                 remove_digits=True,
                 remove_punct=True,
                 remove_stopwords=True,
                 normalize_method='lemma',
                 custom_stopwords=None):

        self.cleaner = TextCleaner()
        self.tokenizer = Tokenizer()
        self.stop_handler = StopWordsHandler(language)
        self.normalizer = WordNormalizer()

        # Configuration
        self.remove_digits = remove_digits
        self.remove_punct = remove_punct
        self.remove_stopwords = remove_stopwords
        self.normalize_method = normalize_method

        if custom_stopwords:
            self.stop_handler.add_custom_stopwords(custom_stopwords)

    def preprocess_single(self, text, return_steps=False):
        """Preprocess a single text with optional step tracking"""
        steps = {}

        # Step 1: Clean text
        cleaned = self.cleaner.clean_text(text, self.remove_digits, self.remove_punct)
        steps['cleaned'] = cleaned

        # Step 2: Advanced cleaning
        advanced_cleaned = self.cleaner.advanced_clean(cleaned)
        steps['advanced_cleaned'] = advanced_cleaned

        # Step 3: Tokenization
        tokens = self.tokenizer.word_tokenize_nltk(advanced_cleaned)
        steps['tokens'] = tokens

        # Step 4: Remove stop words
        if self.remove_stopwords:
            tokens = self.stop_handler.remove_stopwords(tokens)
            steps['no_stopwords'] = tokens

        # Step 5: Normalization
        if self.normalize_method == 'stem':
                tokens = self.normalizer.stem_words(tokens, 'porter')
            elif self.normalize_method == 'lemma':
                pos_tags = pos_tag(tokens)
                pos_only = [pos for token, pos in pos_tags]
                tokens = self.normalizer.lemmatize_words(tokens, pos_only)

            steps['normalized'] = tokens

            if return_steps:
                return tokens, steps
            return tokens

        def preprocess_batch(self, texts, show_progress=True):
            """Preprocess multiple texts"""
            results = []
            total = len(texts)

            for i, text in enumerate(texts):
                if show_progress and (i + 1) % 100 == 0:
                    print(f"Processed {i + 1}/{total} texts")

                processed = self.preprocess_single(text)
                results.append(processed)

            return results

        def get_statistics(self, original_texts, processed_texts):
            """Get preprocessing statistics"""
            stats = {
                'total_texts': len(original_texts),
                'avg_original_length': np.mean([len(text.split()) for text in original_texts]),
                'avg_processed_length': np.mean([len(tokens) for tokens in processed_texts]),
                'total_original_tokens': sum(len(text.split()) for text in original_texts),
                'total_processed_tokens': sum(len(tokens) for tokens in processed_texts)
            }

            stats['reduction_percentage'] = (
                (stats['total_original_tokens'] - stats['total_processed_tokens']) /
                stats['total_original_tokens'] * 100
            )

            return stats

        def save_processed_data(self, texts, processed_texts, filename):
            """Save processed data to file"""
            df = pd.DataFrame({
                'original': texts,
            'processed': [' '.join(tokens) for tokens in processed_texts],
            'token_count': [len(tokens) for tokens in processed_texts]
        })

        df.to_csv(filename, index=False)
        print(f"Processed data saved to {filename}")

# Comprehensive example
preprocessor = TextPreprocessor(
    remove_digits=True,
    remove_punct=True,
    remove_stopwords=True,
    normalize_method='lemma',
    custom_stopwords=['said', 'says', 'think']
)

# Sample texts for testing
sample_texts = [
    "I absolutely LOVE this new restaurant! The food is amazing and the service is excellent.
    "This movie was terrible... I can't believe I wasted 2 hours watching it. Not recommended!"
    "The weather forecast says it'll rain tomorrow. Don't forget your umbrella!",
    "Machine learning and artificial intelligence are revolutionizing the tech industry in 2024
    "Visit our website at https://example.com for more information. Email us at info@example.c
]

print("Complete Text Preprocessing Pipeline:")
print("=" * 60)

# Process single text with step tracking
text = sample_texts[0]
processed_tokens, steps = preprocessor.preprocess_single(text, return_steps=True)

print(f"Original text: {text}")
print("\nProcessing steps:")
for step_name, result in steps.items():
    if isinstance(result, list):
        result_str = ' '.join(result) if result else '[]'
    else:
        result_str = result
    print(f"{step_name}: {result_str}")

print("\n" + "="*60)

# Process all sample texts
all_processed = preprocessor.preprocess_batch(sample_texts, show_progress=False)

# Get statistics
```

```python
stats = preprocessor.get_statistics(sample_texts, all_processed)

print("Preprocessing Statistics:")
print("-" * 30)
for key, value in stats.items():
    if isinstance(value, float):
        print(f"{key.replace('_', ' ').title()}: {value:.2f}")
    else:
        print(f"{key.replace('_', ' ').title()}: {value}")

# Show results for all texts
print("\nAll Processed Texts:")
print("-" * 30)
for i, (original, processed) in enumerate(zip(sample_texts, all_processed), 1):
    print(f"{i}. Original: {original[:60]}...")
    print(f"   Processed: {' '.join(processed)}")
    print()
```

🌐 Web Scraping with Beautiful Soup

Theory: Web Scraping for NLP

Web scraping is essential for NLP because:

- **Data Collection**: Gather text data from websites
- **Real-time Analysis**: Monitor social media, news, reviews
- **Content Aggregation**: Combine information from multiple sources
- **Research**: Academic and market research

Implementation: Comprehensive Web Scraping

```
python
```

```python
import requests
from bs4 import BeautifulSoup
import time
import csv
from urllib.parse import urljoin, urlparse
import json

class WebScraper:
    def __init__(self, delay=1, timeout=10):
        self.delay = delay  # Delay between requests
        self.timeout = timeout
        self.session = requests.Session()

        # Set user agent to avoid blocking
        self.session.headers.update({
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36'
        })

    def get_page_content(self, url):
        """Safely get page content"""
        try:
            response = self.session.get(url, timeout=self.timeout)
            response.raise_for_status()
            return response.content
        except requests.RequestException as e:
            print(f"Error fetching {url}: {e}")
            return None

    def extract_text_content(self, html_content, content_tags=None):
        """Extract text from HTML content"""
        if content_tags is None:
            content_tags = ['p', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'article', 'div']

        soup = BeautifulSoup(html_content, 'html.parser')

        # Remove script and style elements
        for script in soup(["script", "style"]):
            script.decompose()

        # Extract text from specified tags
        texts = []
        for tag in content_tags:
            elements = soup.find_all(tag)
            for element in elements:
                text = element.get_text(strip=True)
                if len(text) > 20:  # Only keep substantial text
                texts.append(text)

        return texts

    def scrape_news_article(self, url):
        """Specialized scraper for news articles"""
        content = self.get_page_content(url)
        if not content:
            return None

        soup = BeautifulSoup(content, 'html.parser')

        # Try to extract title
        title = None
        for selector in ['h1', 'title', '.article-title', '.headline']:
            element = soup.select_one(selector)
            if element:
                title = element.get_text(strip=True)
                break

        # Try to extract article body
        article_text = []
        article_selectors = [
            '.article-body', '.article-content', '.post-content',
            '.entry-content', 'article p', '.story-body p'
        ]

        for selector in article_selectors:
            paragraphs = soup.select(selector)
            if paragraphs:
                for p in paragraphs:
                    text = p
```