Natural Language Processing

An Introduction

Abdelbacet Mhamdi

2025-01-12

ISET Bizerte

Outline

1. Introduction to Regular Expressions (Regex)

Regular expressions are powerful patterns used to match, search, and manipulate text strings. They provide a standardized way to describe search patterns in text, making them an essential tool in programming, text processing, and data validation.

1.2.1 Pattern Matching

A regex pattern is a sequence of characters that defines a search pattern. These patterns can be:

- Literal characters that match themselves
- Special characters (metacharacters) with special meanings
- Combinations of both

1.2.2 Basic Metacharacters

Metacharacter	Description	Example
`.`	Matches any character except newline	`a.c` matches "abc", "a1c", "a@c"
`^`	Matches start of string	`^Hello` matches "Hello World"
`$`	Matches end of string	`world$` matches "Hello world"
`*`	Matches 0 or more occurrences	`ab*c` matches "ac", "abc", "abbc"

Metacharacter	Description	Example
+	Matches 1 or more occurrences	ab+c matches “abc”, “abbc” but not “ac”
?	Matches 0 or 1 occurrence	ab?c matches “ac” and “abc”
\	Escapes special characters	`\.` matches literal dot

1.3 Common Use Cases

1. **Data Validation**
   - Email addresses
   - Phone numbers
   - Postal codes
   - Passwords
   - URLs
2. **Text Processing**
   - Finding patterns in text
   - Replacing specific text patterns
   - Extracting information
   - Parsing log files
3. **Search Operations**
   - Advanced find/replace operations

1.3 Common Use Cases

- Pattern matching in large text files
- Content filtering

1.4.1 Basic Pattern Matching

Python

```python
import re

# Simple pattern matching
text = "The quick brown fox jumps over the lazy dog"
pattern = r"fox"

# Search for pattern
match = re.search(pattern, text)
if match:
    print(f"Found '{pattern}' at position: {match.start()}-{match.end()}")

# Find all occurrences
```

```python
words = re.findall(r"\w+", text)
print(f"All words: {words}")
```

1.4.2 Email Validation Example

Python

```python
def is_valid_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return bool(re.match(pattern, email))

# Test cases
emails = [
    "user@example.com",
    "invalid.email@com",
    "user.name+tag@domain.co.uk",
```

```python
    "@invalid.com"
]

for email in emails:
    print(f"{email}: {'Valid' if is_valid_email(email) else 'Invalid'}")
```

1.4.3 Phone Number Formatting

Python

```python
def format_phone_number(phone):
    # Remove all non-digit characters
    digits = re.sub(r'\D', '', phone)

    # Format as (XXX) XXX-XXXX
    if len(digits) == 10:
```

```python
        pattern = r'(\d{3})(\d{3})(\d{4})'
        formatted = re.sub(pattern, r'(\1) \2-\3', digits)
        return formatted
    return "Invalid phone number"

# Test cases
numbers = [
    "1234567890",
    "123-456-7890",
    "(123) 456-7890",
    "12345"
]

```

```python
for number in numbers:
    print(f"{number} → {format_phone_number(number)}")
```

1.4.4 Advanced Concepts

1.4.5 Character Classes

Python

```python
# Character class examples
pattern = r'[aeiou]'  # Matches any vowel
pattern = r'[0-9]'    # Matches any digit
pattern = r'[^0-9]'   # Matches any non-digit
```

1.4.6 Quantifiers and Groups

Python

```python
# Quantifiers
pattern = r'\d{3}'     # Exactly 3 digits
```

```python
pattern = r'\d{2,4}'    # Between 2 and 4 digits
pattern = r'\d{2,}'     # 2 or more digits

# Groups
pattern = r'(\w+)\s+\1'  # Matches repeated words
```

1.4.7 Common Regex Functions in Python

Python

```python
import re

text = "The price is $19.99"

# Different matching functions
re.search(r'\$\d+\.\d+', text)   # Finds first match
```

```python
re.findall(r'\$\d+\.\d+', text)  # Finds all matches
re.sub(r'\$(\d+\.\d+)', r'\1', text)  # Substitution

# Splitting text
re.split(r'\s+', text)  # Split on whitespace
```

1. **Use Raw Strings**
   - Always prefix regex patterns with `r` to avoid escape character issues

```python
pattern = r'\d+'  # Better than '\d+'
```

Python

2. **Compile Frequently Used Patterns**

```python
email_pattern = re.compile(r'^[\w\.-]+@[\w\.-]+\.\w+$')
# Use multiple times
email_pattern.match(email1)
email_pattern.match(email2)
```

Python

3. **Be Specific**
   - Make patterns as specific as possible to avoid false matches
   - Use start (^) and end ($) anchors when matching whole strings

4. **Test Thoroughly**
    - Test with both valid and invalid inputs
    - Include edge cases in your tests

Python

```python
def test_pattern(pattern, test_cases):
    regex = re.compile(pattern)
    for test, expected in test_cases:
        result = bool(regex.match(test))
        print(f"'{test}': {'✓' if result == expected else '✗'}")
```

1. **Greedy vs. Non-Greedy Matching**

```python
# Greedy (default)
re.findall(r'<.*>', '<tag>text</tag>')  # ['<tag>text</tag>']

# Non-greedy
re.findall(r'<.*?>', '<tag>text</tag>')  # ['<tag>', '</tag>']
```

2. **Performance Considerations**
   - Avoid excessive backtracking
   - Be careful with nested quantifiers
   - Use more specific patterns when possible

1. **Basic Pattern Matching**

```python
# Write a pattern to match dates in format DD/MM/YYYY
date_pattern = r'\d{2}/\d{2}/\d{4}'
```

2. **Data Extraction**

```python
# Extract all email addresses from text
text = "Contact us at support@example.com or sales@example.com"
emails = re.findall(r'[\w\.-]+@[\w\.-]+\.\w+', text)
```

3. **Password Validation**

```python
def is_strong_password(password):
    # At least 8 chars, 1 upper, 1 lower, 1 digit, 1 special
```

1.7 Exercise Examples

```
    pattern = r'^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$'
    return bool(re.match(pattern, password))
```

2. Text Tokenization Guide

2.1 Introduction to Tokenization

Tokenization is the process of breaking down text into smaller units called tokens. These tokens can be words, characters, subwords, or phrases depending on the specific requirements of your NLP task.

2.2.1 Simple Word Tokenization

Python

```python
import re

def simple_word_tokenize(text):
    # Split on whitespace and punctuation
    tokens = re.findall(r'\b\w+\b', text)
    return tokens

text = "Hello, world! This is a simple example."
tokens = simple_word_tokenize(text)
print(tokens)  # ['Hello', 'world', 'This', 'is', 'a', 'simple', 'example']
```

2.2.2 Advanced Regex Tokenization

Python

```python
def advanced_tokenize(text):
    # Pattern matches words, numbers, punctuation, and special characters
    pattern = r"""
        [\w]+                          # Word characters
        |(?:[\$])?\d+(?:\.\d+)?%?  # Numbers with optional decimal points, $, %
        |[.,!?;"]                     # Punctuation
        |[:']                         # Special characters
    """
    tokens = re.findall(pattern, text, re.VERBOSE)
    return tokens

text = "The price is $19.99, and the discount is 15%!"
```

```python
print(advanced_tokenize(text))
# ['The', 'price', 'is', '$19.99', ',', 'and', 'the', 'discount', 'is', '15%', '!']
```

NLTK provides various tokenizers for different needs.

2.3.1 Installation and Setup

```python
import nltk
nltk.download('punkt')  # Required for word and sentence tokenization
```

2.3.2 Word Tokenization

```python
from nltk.tokenize import word_tokenize, TreebankWordTokenizer

# Simple word tokenization
text = "Don't hesitate to use NLTK's tokenizer."
tokens = word_tokenize(text)
print(tokens)
# ["Do", "n't", "hesitate", "to", "use", "NLTK", "'s", "tokenizer", "."]
```

```python

# TreebankWordTokenizer for Penn Treebank style tokenization
treebank = TreebankWordTokenizer()
tokens = treebank.tokenize(text)
print(tokens)
```

2.3.3 Sentence Tokenization

Python

```python
from nltk.tokenize import sent_tokenize

text = """Mr. Smith bought a car. He loves driving it!
          What will he buy next? Only time will tell."""
sentences = sent_tokenize(text)
print(sentences)
```

2.3.4 Regular Expression Tokenizer

Python

```python
from nltk.tokenize import RegexpTokenizer

# Create custom patterns
tokenizer = RegexpTokenizer(r'\w+|[^\w\s]+')
text = "Hello, World! How's it going?"
tokens = tokenizer.tokenize(text)
print(tokens)
```

spaCy provides more advanced tokenization with linguistic features.

2.4.1 Installation and Setup

Python

```python
import spacy
nlp = spacy.load('en_core_web_sm')
```

2.4.2 Basic Tokenization

Python

```python
def spacy_tokenize(text):
    doc = nlp(text)
    return [token.text for token in doc]

text = "spaCy's tokenizer is industrial-strength!"
tokens = spacy_tokenize(text)
print(tokens)
```

```
# ["spaCy", "'s", "tokenizer", "is", "industrial", "-", "strength", "!"]
```

2.4.3 Advanced Features

Python

```python
def analyze_tokens(text):
    doc = nlp(text)
    for token in doc:
        print(f"""
        Text: {token.text}
        Lemma: {token.lemma_}
        POS: {token.pos_}
        Is stop word: {token.is_stop}
        """)

```

2.4 3. spaCy Tokenization

```python
text = "Running quickly through the forest"
analyze_tokens(text)
```

Polyglot is especially useful for multilingual tokenization.

2.5.1 Installation and Setup

Python

```python
from polyglot.text import Text
```

2.5.2 Basic Usage

Python

```python
def polyglot_tokenize(text):
        text = Text(text)
        return list(text.words)

# Multiple languages
english_text = "Hello, world!"
spanish_text = "¡Hola, mundo!"
chinese_text = "你好，世界！"
```

```python

for sample in [english_text, spanish_text, chinese_text]:
    tokens = polyglot_tokenize(sample)
    print(f"Original: {sample}")
    print(f"Tokens: {tokens}\n")
```

2.6.1 Handling Special Cases

```python
text = "Don't forget the U.S.A.'s high-quality standards!"

# Compare different tokenizers
print("Regex:", simple_word_tokenize(text))
print("NLTK:", word_tokenize(text))
print("spaCy:", spacy_tokenize(text))
print("Polyglot:", polyglot_tokenize(text))
```

2.6.2 Strengths and Use Cases

1. **Regex-based Tokenization**
   - Best for: Simple, custom tokenization rules
   - Pros: Fast, flexible, easy to modify
   - Cons: Can’t handle complex linguistic cases

2. **NLTK**
    - Best for: Academic and research projects
    - Pros: Rich features, well-documented
    - Cons: Slower than some alternatives
3. **spaCy**
    - Best for: Production environments
    - Pros: Fast, modern, good defaults
    - Cons: Larger memory footprint
4. **Polyglot**
    - Best for: Multilingual projects
    - Pros: Excellent language coverage
    - Cons: Can be slower, fewer features

1. **Choose the Right Tokenizer**

```python
def select_tokenizer(text, language='en', needs_speed=False):
    if needs_speed and language == 'en':
        return spacy_tokenize(text)
    elif language != 'en':
        return polyglot_tokenize(text)
    else:
        return word_tokenize(text)
```

2. **Pre-processing**

```python
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
```

```
    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()
    # Remove special characters (if needed)
    text = re.sub(r'[^\w\s]', '', text)
    return text
```

3. **Handling Special Cases**

```python
def handle_special_cases(tokens):
    # Handle contractions
    contractions = {"n't": "not", "'s": "is"}
    expanded_tokens = []
    for token in tokens:
        if token in contractions:
```

```
                expanded_tokens.append(contractions[token])
            else:
                expanded_tokens.append(token)
        return expanded_tokens
```

3. Text Processing, Visualization, and Concepts Guide

3.1.1 What is Text Preprocessing?

Text preprocessing is the process of cleaning and transforming raw text into a format that's more suitable for analysis. Think of it as preparing ingredients before cooking - just as you wash and chop vegetables before cooking, you clean and standardize text before analysis.

Real-world example:

```
Raw text: "RT @username: Check out our new product!!! It's AMAZING... 😊 www.example.com #awesome"
Preprocessed text: "check out our new product it is amazing"
```

3.1.2 Key Preprocessing Steps

3.1.2.1 1. Tokenization

Definition: The process of breaking down text into individual units (tokens), typically words or subwords.

Real-world examples:

```
Sentence: "I love natural language processing!"
Tokens: ["I", "love", "natural", "language", "processing", "!"]

Sentence: "New York City is beautiful"
Tokens: ["New", "York", "City", "is", "beautiful"]
```

3.1.2.2 2. Stop Word Removal

Definition: Eliminating common words that typically don't carry significant meaning.

Common stop words in English: "the", "is", "at", "which", "on", etc.

Real-world example:

```
Original: "The cat is on the mat"
```

```
After stop word removal: "cat mat"
```

3.1.2.3 3. Lemmatization

Definition: Reducing words to their base or dictionary form (lemma).

Real-world examples:

```
am, are, is → be
running, ran, runs → run
better, best → good
wolves → wolf
```

3.1.2.4 4. Stemming

Definition: Reducing words to their root form by removing affixes, often resulting in non-dictionary words.

Real-world examples:

```
running → run
fishing → fish
completely → complet
authentication → authent
```

3.1.3 Bag of Words (BoW)

Definition: A text representation method that describes the occurrence of words within a document. It creates a vocabulary of unique words and represents each document as a vector of word frequencies.

Real-world example:

```
Documents:
```

```
1. "The cat likes milk"
2. "The dog hates milk"

Vocabulary: ["the", "cat", "dog", "likes", "hates", "milk"]
BoW representations:
Doc 1: [1, 1, 0, 1, 0, 1]
Doc 2: [1, 0, 1, 0, 1, 1]
```

3.1.4 Term Frequency-Inverse Document Frequency (TF-IDF)

Definition: A numerical statistic that reflects how important a word is to a document in a collection of documents.

Real-world example:

```
Consider these news articles:
```

3.1 Fundamental Concepts and Definitions

```
1. "The new iPhone features advanced AI capabilities"
2. "The new Android phone launches today"
3. "The weather is nice today"

The word "the" appears in all documents, so it gets a low IDF score.
The word "iPhone" appears in only one document, so it gets a high IDF score.
```

3.2.1 Word Clouds

Definition: A visual representation where word size corresponds to its frequency in the text.

Real-world applications:

- Analyzing customer reviews to identify common themes
- Visualizing key topics in political speeches
- Summarizing survey responses

3.2.2 Frequency Distribution Plots

Definition: Charts showing how often different words appear in a text.

Real-world applications:

- Comparing vocabulary usage across different authors
- Analyzing Twitter hashtag popularity over time
- Studying language patterns in different genres of literature

Let's analyze a customer review:

Original review:

```
"I've been using this phone for 3 months now... It's AMAZING!!! The battery life
is incredible, and the camera takes beautiful pics. Can't believe how good it is
:) Would definitely recommend to my friends & family!!!"
```

Pipeline steps:

1. **Cleaning**:

   ```
   "i have been using this phone for three months now it is amazing the battery
   ```

3.3 Complete Pipeline Example with Real Text

```
life is incredible and the camera takes beautiful pictures cannot believe how
good it is would definitely recommend to my friends and family"
```

2. **Tokenization**:

```
["i", "have", "been", "using", "this", "phone", "for", "three", "months", ...]
```

3. **Stop Word Removal**:

```
["phone", "three", "months", "amazing", "battery", "life", "incredible",
"camera", "takes", "beautiful", "pictures", "good", "definitely", "recommend",
"friends", "family"]
```

4. **Lemmatization**:

```
["phone", "month", "amazing", "battery", "life", "incredible", "camera",
"take", "beautiful", "picture", "good", "definitely", "recommend", "friend",
"family"]
```

3.4 Common Use Cases and Applications

1. **Sentiment Analysis**
   - Customer review processing
   - Social media monitoring
   - Brand reputation tracking
2. **Content Classification**
   - News article categorization
   - Spam detection
   - Document sorting
3. **Text Summarization**
   - News article summarization
   - Document abstract generation
   - Meeting notes condensation
4. **Keyword Extraction**

3.4 Common Use Cases and Applications

- SEO optimization
- Content tagging
- Research paper indexingA

4. Text Processing and Visualization Guide

4.1.1 Basic Word Frequency Chart

Python

```python
import matplotlib.pyplot as plt
from collections import Counter
import seaborn as sns

def plot_word_frequency(text, top_n=10):
    # Tokenize and count words
    words = text.lower().split()
    word_freq = Counter(words)

    # Get top N words
    top_words = dict(word_freq.most_common(top_n))

```

```python
    # Create bar plot
    plt.figure(figsize=(12, 6))
    plt.bar(top_words.keys(), top_words.values())
    plt.xticks(rotation=45, ha='right')
    plt.title('Top Word Frequencies')
    plt.xlabel('Words')
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.show()
```

4.1.2 Word Cloud Visualization

Python

```python
from wordcloud import WordCloud

```

```python
def generate_wordcloud(text):
    wordcloud = WordCloud(
        width=800, height=400,
        background_color='white',
        max_words=100
    ).generate(text)

    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.show()
```

4.1.3 Advanced Visualization with Seaborn

Python

```python
def plot_word_distribution(text, top_n=10):
    # Create word frequency distribution
    words = text.lower().split()
    word_freq = Counter(words)

    # Convert to DataFrame for Seaborn
    import pandas as pd
    df = pd.DataFrame(word_freq.most_common(top_n),
                      columns=['Word', 'Frequency'])

    # Create plot
    plt.figure(figsize=(12, 6))
```

4.1 1. Word Frequency Visualization

```
    sns.barplot(data=df, x='Word', y='Frequency')
    plt.xticks(rotation=45, ha='right')
    plt.title('Word Frequency Distribution')
    plt.tight_layout()
    plt.show()
```

4.2.1 Simple Bag of Words

Python

```python
from sklearn.feature_extraction.text import CountVectorizer

def create_bow(documents):
    # Initialize vectorizer
    vectorizer = CountVectorizer()

    # Create BOW representation
    X = vectorizer.fit_transform(documents)

    # Get feature names (vocabulary)
    feature_names = vectorizer.get_feature_names_out()

```

```python
    # Convert to array
    bow_array = X.toarray()

    return bow_array, feature_names, vectorizer

# Example usage
documents = [
    "The cat sat on the mat",
    "The dog ran in the park",
    "The cat and dog played"
]

bow_array, features, vectorizer = create_bow(documents)
```

```python
print("Features:", features)
print("BOW Matrix:\n", bow_array)
```

4.2.2 TF-IDF Implementation

Python

```python
from sklearn.feature_extraction.text import TfidfVectorizer

def create_tfidf(documents):
    # Initialize TF-IDF vectorizer
    tfidf = TfidfVectorizer()

    # Create TF-IDF matrix
    X = tfidf.fit_transform(documents)

```

```python
    # Get feature names
    feature_names = tfidf.get_feature_names_out()

    return X.toarray(), feature_names, tfidf
```

4.3.1 Comprehensive Pipeline Class

Python

```python
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer

class TextPreprocessor:
    def __init__(self, language='english'):
        # Download required NLTK data
        nltk.download('punkt')
        nltk.download('stopwords')
```

4.3 3. Complete Text Preprocessing Pipeline

```python
        nltk.download('wordnet')

        self.language = language
        self.stop_words = set(stopwords.words(language))
        self.lemmatizer = WordNetLemmatizer()
        self.stemmer = PorterStemmer()

    def clean_text(self, text):
        """Basic text cleaning"""
        # Convert to lowercase
        text = text.lower()

        # Remove special characters and digits
```

```python
        text = re.sub(r'[^a-zA-Z\s]', '', text)

        # Remove extra whitespace
        text = re.sub(r'\s+', ' ', text).strip()

        return text

    def tokenize(self, text):
        """Tokenize text"""
        return word_tokenize(text)

    def remove_stopwords(self, tokens):
        """Remove stop words"""
```

4.3 3. Complete Text Preprocessing Pipeline

```python
        return [token for token in tokens if token not in self.stop_words]

    def lemmatize(self, tokens):
        """Lemmatize tokens"""
        return [self.lemmatizer.lemmatize(token) for token in tokens]

    def stem(self, tokens):
        """Stem tokens"""
        return [self.stemmer.stem(token) for token in tokens]

    def process(self, text, use_stemming=False):
        """Complete preprocessing pipeline"""
        # Clean text
```

```python
        cleaned_text = self.clean_text(text)

        # Tokenize
        tokens = self.tokenize(cleaned_text)

        # Remove stopwords
        tokens = self.remove_stopwords(tokens)

        # Lemmatize or stem
        if use_stemming:
            tokens = self.stem(tokens)
        else:
            tokens = self.lemmatize(tokens)
```

```python

        return tokens

# Example usage
preprocessor = TextPreprocessor()
text = "The cats are running quickly through the forest!"
processed_tokens = preprocessor.process(text)
print("Processed tokens:", processed_tokens)
```

4.3.2 Advanced Pipeline with Custom Features

Python

```python
class AdvancedTextPreprocessor(TextPreprocessor):
    def __init__(self, language='english', custom_stopwords=None):
        super().__init__(language)
```

```python

        # Add custom stopwords if provided
        if custom_stopwords:
            self.stop_words.update(custom_stopwords)

    def remove_short_words(self, tokens, min_length=3):
        """Remove words shorter than min_length"""
        return [token for token in tokens if len(token) >= min_length]

    def normalize_elongated_words(self, text):
        """Normalize elongated words (e.g., 'hellooo' -> 'hello')"""
        pattern = re.compile(r'(.)\1{2,}')
        return pattern.sub(r'\1\1', text)
```

```python

    def process(self, text, use_stemming=False, min_word_length=3):
        # Normalize elongated words
        text = self.normalize_elongated_words(text)

        # Get tokens from parent class
        tokens = super().process(text, use_stemming)

        # Remove short words
        tokens = self.remove_short_words(tokens, min_word_length)

        return tokens
```

4.4 4. Complete Example Pipeline

Python

```python
def process_and_visualize_text(text):
    # Initialize preprocessor
    preprocessor = AdvancedTextPreprocessor()

    # Process text
    tokens = preprocessor.process(text)

    # Create BOW representation
    bow_array, features, _ = create_bow([' '.join(tokens)])

    # Create visualizations
    print("Processed tokens:", tokens)
    print("\nBag of Words representation:")
```

```python
    print("Features:", features)
    print("BOW array:", bow_array)

    # Generate word frequency plot
    plot_word_frequency(' '.join(tokens))

    # Generate word cloud
    generate_wordcloud(' '.join(tokens))

    return tokens, bow_array, features

# Example usage
sample_text = """
```

4.4 4. Complete Example Pipeline

```python
Natural language processing (NLP) is a subfield of linguistics, computer science,
and artificial intelligence concerned with the interactions between computers and
human language, in particular how to program computers to process and analyze large
amounts of natural language data.
"""

tokens, bow, features = process_and_visualize_text(sample_text)
```

1. **Choose the Right Tools**
   - Use NLTK for research and experimentation
   - Use spaCy for production environments
   - Use scikit-learn for machine learning integration
2. **Performance Optimization**

Python

```python
# Cache processed results for large datasets
from functools import lru_cache

@lru_cache(maxsize=1000)
def cached_preprocess(text):
    preprocessor = TextPreprocessor()
    return preprocessor.process(text)
```

3. **Error Handling**

```python
def safe_preprocess(text):
    try:
        return preprocessor.process(text)
    except Exception as e:
        print(f"Error processing text: {e}")
        return []
```

4. **Evaluation Metrics**

```python
def evaluate_preprocessing(original_text, processed_tokens):
    # Calculate reduction ratio
    original_tokens = word_tokenize(original_text)
    reduction_ratio = 1 - (len(processed_tokens) / len(original_tokens))
```

```python

    print(f"Original token count: {len(original_tokens)}")
    print(f"Processed token count: {len(processed_tokens)}")
    print(f"Reduction ratio: {reduction_ratio:.2%}")
```

5. Gensim Text Processing Guide

5.1 What is Gensim?

Gensim is a robust, efficient library for topic modeling, document indexing, and similarity retrieval with large corpora. The name “Gensim” stands for “Generate Similar” - reflecting its core functionality of finding similar documents.

Key features:

- Memory efficient processing of large text collections
- Built-in implementations of popular algorithms like Word2Vec, Doc2Vec, FastText
- Streamlined document similarity calculations
- Topic modeling capabilities (LSA, LDA)

5.2.1 Installation and Setup

Python

```python
pip install gensim
import gensim
from gensim import corpora, models
```

5.2.2 Creating a Document Corpus

Python

```python
# Sample documents
documents = [
    "The quick brown fox jumps over the lazy dog",
    "Python is a great programming language",
    "Text processing with Gensim is efficient",
    "The lazy dog sleeps all day",
]
```

```python

# Tokenize documents
def preprocess(text):
    # Convert to lowercase and split into words
    return text.lower().split()

# Process all documents
processed_docs = [preprocess(doc) for doc in documents]

# Create dictionary (maps words to IDs)
dictionary = corpora.Dictionary(processed_docs)

# Convert documents to bag-of-words format
```

```python
corpus = [dictionary.doc2bow(doc) for doc in processed_docs]

print("Dictionary:", dictionary.token2id)
print("\nFirst document BoW:", corpus[0])
```

5.3.1 Understanding TF-IDF in Gensim

TF-IDF (Term Frequency-Inverse Document Frequency) in Gensim helps identify important words by considering both their frequency in individual documents and their rarity across all documents.

Real-world example:

```
Consider a collection of news articles:
- Common words like "the" or "and" appear frequently but in most documents
- Topic-specific words like "cryptocurrency" might appear less frequently but in specific documents
- TF-IDF will give higher weights to topic-specific words
```

5.3.2 Basic TF-IDF Implementation

```python
from gensim import models
```

```python

# Create TF-IDF model
tfidf = models.TfidfModel(corpus)

# Transform corpus to TF-IDF space
corpus_tfidf = tfidf[corpus]

# Print TF-IDF vectors for each document
for doc in corpus_tfidf:
    print("\nTF-IDF scores:", doc)
```

5.3.3 Advanced TF-IDF Processing

Python

```python
def create_tfidf_model(documents):
```

```python
    """
    Create a TF-IDF model from a list of documents
    """
    # Preprocess documents
    processed_docs = [preprocess(doc) for doc in documents]

    # Create dictionary
    dictionary = corpora.Dictionary(processed_docs)

    # Create BOW corpus
    bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]

    # Create TF-IDF model
```

```python
    tfidf_model = models.TfidfModel(bow_corpus)

    # Transform corpus to TF-IDF space
    corpus_tfidf = tfidf_model[bow_corpus]

    return dictionary, tfidf_model, corpus_tfidf

# Example usage
docs = [
    "Machine learning is fascinating",
    "Deep learning is a subset of machine learning",
    "Neural networks are used in deep learning",
    "Python is great for machine learning"
```

```python
]

dictionary, tfidf_model, corpus_tfidf = create_tfidf_model(docs)

# Print TF-IDF scores for each document
for i, doc in enumerate(corpus_tfidf):
    print(f"\nDocument {i+1} TF-IDF scores:")
    for id, score in doc:
        print(f"Word: {dictionary[id]}, Score: {score:.4f}")
```

5.4.1 Computing Similarity Between Documents

Python

```python
from gensim import similarities

def compute_document_similarity(documents, query):
    """
    Compute similarity between a query and all documents
    """
    # Create TF-IDF model
    dictionary, tfidf_model, corpus_tfidf = create_tfidf_model(documents)

    # Convert query to TF-IDF space
    query_bow = dictionary.doc2bow(preprocess(query))
    query_tfidf = tfidf_model[query_bow]
```

5.4 3. Document Similarity with TF-IDF

```python

    # Initialize similarity matrix
    index = similarities.MatrixSimilarity(corpus_tfidf)

    # Compute similarities
    sims = index[query_tfidf]

    # Return sorted similarities
    return list(enumerate(sims))

# Example usage
documents = [
    "The cat sits on the mat",
```

```python
    "The dog runs in the park",
    "Cats and dogs are pets",
    "The mat is comfortable"
]

query = "Where is the cat sitting?"
similarities = compute_document_similarity(documents, query)

# Print sorted similarities
print("\nDocument similarities to query:")
for doc_id, score in sorted(similarities, key=lambda x: x[1], reverse=True):
    print(f"Document {doc_id+1}: {score:.4f} - {documents[doc_id]}")
```

5.5.1 Building a Complete Text Analysis Pipeline

Python

```python
class GensimTextAnalyzer:
    def __init__(self):
        self.dictionary = None
        self.tfidf_model = None
        self.similarity_index = None

    def fit(self, documents):
        """Train the analyzer on a corpus of documents"""
        # Preprocess documents
        processed_docs = [preprocess(doc) for doc in documents]

        # Create dictionary
```

```
        self.dictionary = corpora.Dictionary(processed_docs)

        # Create BOW corpus
        bow_corpus = [self.dictionary.doc2bow(doc) for doc in
        processed_docs]

        # Create TF-IDF model
        self.tfidf_model = models.TfidfModel(bow_corpus)

        # Transform corpus to TF-IDF space
        corpus_tfidf = self.tfidf_model[bow_corpus]

        # Create similarity index
```

```python
        self.similarity_index = similarities.MatrixSimilarity(corpus_tfidf)

    def get_similar_documents(self, query, top_n=5):
        """Find most similar documents to query"""
        # Process query
        query_bow = self.dictionary.doc2bow(preprocess(query))
        query_tfidf = self.tfidf_model[query_bow]

        # Compute similarities
        sims = self.similarity_index[query_tfidf]

        # Return top N similar documents
```

5.5 4. Advanced Gensim Features

```python
        return sorted(enumerate(sims), key=lambda x: x[1], reverse=True)[:top_n]

# Example usage
analyzer = GensimTextAnalyzer()

documents = [
    "Artificial intelligence is transforming industries",
    "Machine learning models need good data",
    "Deep learning requires powerful GPUs",
    "Data science combines statistics and programming",
    "Neural networks are inspired by biology"
]

```

```python
# Train analyzer
analyzer.fit(documents)

# Find similar documents
query = "How is AI changing the world?"
similar_docs = analyzer.get_similar_documents(query)

print("\nMost similar documents to query:")
for doc_id, score in similar_docs:
    print(f"Score: {score:.4f} - {documents[doc_id]}")
```

1. **Memory Efficiency**
   - Use streaming corpus for large datasets
   - Implement memory-efficient iterators

```python
class MyCorpus:
    def __iter__(self):
        for line in open('mycorpus.txt'):
            yield dictionary.doc2bow(line.lower().split())
```

2. **Preprocessing**
   - Remove stop words
   - Apply lemmatization
   - Handle special characters

```python
def advanced_preprocess(text):
```

```
    # Remove special characters
    text = re.sub(r'[^\w\s]', '', text)
    # Convert to lowercase
    text = text.lower()
    # Remove stop words
    stop_words = set(['the', 'is', 'at', 'which'])
    return [word for word in text.split() if word not in stop_words]
```

3. **Model Persistence**

```python
# Save models
dictionary.save('dictionary.gensim')
tfidf_model.save('tfidf.gensim')

```

```python
# Load models
dictionary = corpora.Dictionary.load('dictionary.gensim')
tfidf_model = models.TfidfModel.load('tfidf.gensim')
```

6. Named Entity Recognition (NER) Guide

Named Entity Recognition (NER) is a natural language processing technique that identifies and classifies named entities (key elements) in text into predefined categories such as:

- Person names (e.g., "Barack Obama", "Shakespeare")
- Organizations (e.g., "Microsoft", "United Nations")
- Locations (e.g., "Paris", "Mount Everest")
- Date/Time expressions (e.g., "June 2024", "last Monday")
- Monetary values (e.g., "$1000", "€50")
- Percentages (e.g., "25%", "three-quarters")

Real-world example:

```
Input text: "Apple CEO Tim Cook announced new iPhone models in California last September."

Identified entities:
```

```
4  - Apple (ORGANIZATION)
5  - Tim Cook (PERSON)
6  - California (LOCATION)
7  - September (DATE)
```

6.2.1 Basic NER with NLTK

Python

```python
import nltk
from nltk import ne_chunk
from nltk import word_tokenize, pos_tag

# Download required NLTK data
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')

def nltk_ner(text):
    # Tokenize and tag parts of speech
    tokens = word_tokenize(text)
```

```python
    pos_tags = pos_tag(tokens)

    # Perform NER
    named_entities = ne_chunk(pos_tags)

    return named_entities

# Example usage
text = "John works at Google in New York."
entities = nltk_ner(text)
print(entities)
```

6.2.2 Extracting Named Entities

Python

```python
def extract_entities(text):
    """
    Extract and categorize named entities from text
    """
    entities = {
        'PERSON': [],
        'ORGANIZATION': [],
        'GPE': [],  # Geo-Political Entity
        'LOCATION': [],
        'DATE': [],
        'TIME': [],
        'MONEY': [],
```

```
        'PERCENT': []
    }

    # Get named entities
    named_entities = nltk_ner(text)

    # Extract entities
    for chunk in named_entities:
        if hasattr(chunk, 'label'):
            entity_name = ' '.join(c[0] for c in chunk)
            entity_type = chunk.label()
            if entity_type in entities:
                entities[entity_type].append(entity_name)
```

```python

    return entities

# Example usage
text = """
Tim Cook, CEO of Apple Inc., announced yesterday that the company's revenue
grew by 15% to reach $365 billion in New York City.
"""

entities = extract_entities(text)
for entity_type, entity_list in entities.items():
    if entity_list:
        print(f"{entity_type}: {entity_list}")
```

6.3.1 Basic NER with spaCy

Python

```python
import spacy

# Load English language model
nlp = spacy.load("en_core_web_sm")

def spacy_ner(text):
    """
    Perform NER using spaCy
    """
    # Process text
    doc = nlp(text)

```

```python
    # Extract entities
    entities = [
        {
            'text': ent.text,
            'label': ent.label_,
            'start': ent.start_char,
            'end': ent.end_char
        }
        for ent in doc.ents
    ]

    return entities

```

```python
# Example usage
text = "Microsoft's CEO Satya Nadella visited London last week."
entities = spacy_ner(text)

for entity in entities:
    print(f"Entity: {entity['text']}")
    print(f"Type: {entity['label']}")
    print(f"Position: {entity['start']}-{entity['end']}\n")
```

6.3.2 Advanced spaCy NER

```python
class NamedEntityExtractor:
    def __init__(self, model="en_core_web_sm"):
        self.nlp = spacy.load(model)
```

```python

    def analyze_text(self, text):
        """
        Comprehensive NER analysis
        """
        doc = self.nlp(text)

        # Extract entities with context
        analysis = []
        for ent in doc.ents:
            # Get entity context (surrounding words)
            start = max(0, ent.start - 2)
            end = min(len(doc), ent.end + 2)
```

```
            context = doc[start:end].text

            analysis.append({
                'entity': ent.text,
                'type': ent.label_,
                'context': context,
                'explanation': spacy.explain(ent.label_)
            })

        return analysis

    def get_entity_statistics(self, text):
        """
```

```
        Generate statistics about entities in text
        """
        doc = self.nlp(text)

        stats = {
            'total_entities': len(doc.ents),
            'entity_types': {},
            'entity_density': len(doc.ents) / len(doc) if len(doc) > 0 else 0
        }

        # Count entity types
        for ent in doc.ents:
            stats['entity_types'][ent.label_] = \
```

6.3 2. spaCy Implementation

```python
                stats['entity_types'].get(ent.label_, 0) + 1

        return stats

# Example usage
extractor = NamedEntityExtractor()

text = """
In 2024, Google and Microsoft announced a partnership worth $5 billion.
The deal was signed in Seattle by Sundar Pichai and Satya Nadella.
"""

# Analyze text
```

```python
analysis = extractor.analyze_text(text)
print("Named Entity Analysis:")
for item in analysis:
    print(f"\nEntity: {item['entity']}")
    print(f"Type: {item['type']} ({item['explanation']})")
    print(f"Context: \"{item['context']}\"")

# Get statistics
stats = extractor.get_entity_statistics(text)
print("\nEntity Statistics:")
print(f"Total entities found: {stats['total_entities']}")
print("Entity types distribution:", stats['entity_types'])
print(f"Entity density: {stats['entity_density']:.2%}")
```

6.4.1 Training a Custom Model with spaCy

Python

```python
from spacy.tokens import DocBin
from spacy.util import minibatch, compounding

def train_custom_ner(training_data, model=None, output_dir=None, n_iter=100):
    """
    Train a custom NER model

    training_data format:
    [
        ("Text goes here", {"entities": [(0, 4, "LABEL")]}),
        ...
    ]
```

```python
    """
    if model is not None:
        nlp = spacy.load(model)
    else:
        nlp = spacy.blank("en")

    # Create or get NER component
    if "ner" not in nlp.pipe_names:
        ner = nlp.create_pipe("ner")
        nlp.add_pipe("ner")
    else:
        ner = nlp.get_pipe("ner")

```

```python
    # Add labels
    for _, annotations in training_data:
        for ent in annotations.get("entities"):
            ner.add_label(ent[2])

    # Train
    optimizer = nlp.begin_training()
    for itn in range(n_iter):
        losses = {}
        batches = minibatch(training_data, size=compounding(4., 32., 1.001))
        for batch in batches:
            texts, annotations = zip(*batch)
            nlp.update(texts, annotations, drop=0.5, losses=losses)
```

```
            print(f"Loss: {losses}")

    # Save model
    if output_dir is not None:
        nlp.to_disk(output_dir)

    return nlp

# Example training data
training_data = [
    ("Apple Inc. is looking to buy U.K. startup for $1 billion",
     {"entities": [(0, 9, "ORG"), (27, 31, "GPE"), (43, 54, "MONEY")]}),
    ("Microsoft hired new CEO",
```

```
    {"entities": [(0, 9, "ORG")]}),
]

# Train model
custom_model = train_custom_ner(training_data, output_dir="custom_ner_model")
```

6.5.1 1. Text Preprocessing for NER

Python

```python
def preprocess_for_ner(text):
    """
    Preprocess text for better NER results
    """
    # Convert to proper case (helps with name recognition)
    text = text.title()

    # Handle special characters
    text = re.sub(r'[^\w\s.,!?-]', ' ', text)

    # Normalize whitespace
    text = ' '.join(text.split())
```

```python

        return text
```

6.5.2 2. Entity Validation

Python

```python
def validate_entities(entities, gazetteer):
    """
    Validate extracted entities against known lists
    """
    validated_entities = []

    for entity in entities:
        # Check against known entities
        if entity['text'] in gazetteer.get(entity['label'], []):
```

```python
                entity['validated'] = True
            else:
                entity['validated'] = False
            validated_entities.append(entity)

    return validated_entities
```

6.5.3 3. Performance Evaluation

Python

```python
def evaluate_ner_model(model, test_data):
    """
    Evaluate NER model performance
    """
    true_positives = 0
```

```python
        false_positives = 0
        false_negatives = 0

        for text, annotations in test_data:
            # Get predicted entities
            doc = model(text)
            predicted_entities = set([
                (ent.text, ent.label_) for ent in doc.ents
            ])

            # Get true entities
            true_entities = set([
                (text[start:end], label)
```

```
            for start, end, label in annotations['entities']
        ])

        # Calculate metrics
        true_positives += len(predicted_entities & true_entities)
        false_positives += len(predicted_entities - true_entities)
        false_negatives += len(true_entities - predicted_entities)

    # Calculate precision, recall, F1
    precision = true_positives / (true_positives + false_positives)
    recall = true_positives / (true_positives + false_negatives)
    f1 = 2 * (precision * recall) / (precision + recall)

```

```python
    return {
        'precision': precision,
        'recall': recall,
        'f1': f1
    }
```

6.6 5. Common Use Cases

1. **Information Extraction**
   - Extracting company names from news articles
   - Identifying people mentioned in social media posts
   - Finding locations in travel blogs
2. **Document Classification**
   - Categorizing documents based on mentioned organizations
   - Sorting news articles by location
   - Grouping documents by date mentions
3. **Relationship Extraction**
   - Identifying business relationships between companies
   - Finding connections between people
   - Mapping event locations and dates
4. **Content Enrichment**

- Adding metadata to documents
- Linking entities to knowledge bases
- Creating document summaries

Thank you for your attention!