

## ✓ Introduction to Natural Language Processing and Classical Language Models


 Open in Colab

### ✓ Part 1:- Introduction to Basic NLP Operations

- Tokenization
- Normalization
- Stopwords Removal
- Stemming and Lemmatization
- Representing Text:
  - Bag of Words (BoW)
  - Term-Frequency Inverse Term Frequency (TF/IDF)


### ✓ Download NLTK from the Internet and install it on our PC

```
1 ! pip install nltk
```

 Requirement already satisfied: nltk in /home/me/myenv/lib/python3.12/site-packages (3.9.1)  
 Requirement already satisfied: click in /home/me/myenv/lib/python3.12/site-packages (from nltk) (8.1.8)  
 Requirement already satisfied: joblib in /home/me/myenv/lib/python3.12/site-packages (from nltk) (1.4.2)  
 Requirement already satisfied: regex<=2021.8.3 in /home/me/myenv/lib/python3.12/site-packages (from nltk) (2024.11.6)  
 Requirement already satisfied: tqdm in /home/me/myenv/lib/python3.12/site-packages (from nltk) (4.67.1)

### ✓ Download other necessary libraries

```
1 import nltk
2
3 nltk.download('punkt_tab')
4 nltk.download('stopwords') # stopwords are common words that are often removed from text as they are not useful for analysis
5 nltk.download('wordnet') # for nltk.stem using WordNetLemmatizer
```


 [nltk\_data] Downloading package punkt\_tab to /home/me/nltk\_data...  
 [nltk\_data] Package punkt\_tab is already up-to-date!  
 [nltk\_data] Downloading package stopwords to /home/me/nltk\_data...  
 [nltk\_data] Package stopwords is already up-to-date!  
 [nltk\_data] Downloading package wordnet to /home/me/nltk\_data...  
 [nltk\_data] Package wordnet is already up-to-date!  
 True

### ✓ Import NLTK so that we can use it in our code

```
1 import nltk
```

### ✓ Tokenize words using NLTK package

```
1 from nltk.tokenize import word_tokenize
2
3 # punkt_tab is a pretrained tokenization model used for splitting text into sentences and words.
4
5 sentence = "Large language models are revolutionizing business applications."
6 tokens = word_tokenize(sentence)
7 print(tokens)
```

 ['Large', 'language', 'models', 'are', 'revolutionizing', 'business', 'applications', '.']

### ✓ Normalization: Converting text to a standard form to reduce variability:

### ✓ Convert all letters to lower case

```
1 normalized_tokens = [token.lower() for token in tokens]
2 print(normalized_tokens)
```

```
['large', 'language', 'models', 'are', 'revolutionizing', 'business', 'applications', '.']
```

#### Remove punctuation

```
1 import re
2
3 # [^\w\s] means any character that is not a word character or whitespace, ^ inside square brackets negates the expression.
4 # \w is a word character (alphanumeric character plus underscore)
5 normalized_tokens = [re.sub(r'[^\w\s]', '', token.lower()) for token in tokens]
6 print(normalized_tokens)
```

```
['large', 'language', 'models', 'are', 'revolutionizing', 'business', 'applications', '']
```

[Click here to open regular expressions tutorial](#)

#### Stopword Removal:

Eliminating common words that add little meaning

First, we need to download the stopwords dataset and import it in our code

```
1 from nltk.corpus import stopwords
2
3 stop_words = set(stopwords.words('english'))
4
5 print("These words are removed from the text: ", stop_words)
```

```
These words are removed from the text: {'d', 'why', 'you've', 'have', 'hadn', 'again', 'by', 'her', 'is', 'she'll', 'each',
```

Then we can remove stopwords from our sentence

```
1 filtered_tokens = [token for token in normalized_tokens if token and token not in stop_words]
2 print(filtered_tokens)
```

```
['large', 'language', 'models', 'revolutionizing', 'business', 'applications']
```

#### Stemming vs. Lemmatization in NLP

Stemming Example

```
1 from nltk.stem import PorterStemmer
2
3 stemmer = PorterStemmer()
4 stemmed_tokens = [stemmer.stem(token) for token in filtered_tokens]
5
6 print(stemmed_tokens)
```

```
['larg', 'languag', 'model', 'revolution', 'busi', 'applic']
```

Lemmatization Example

```
1 from nltk.stem import WordNetLemmatizer
2
3 lemmatizer = WordNetLemmatizer()
4 lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens]
5
6 print(lemmatized_tokens)
```

```
['large', 'language', 'model', 'revolutionizing', 'business', 'application']
```

## ✓ Representing Text - Bag of Words and TF-IDF

### Bag of Words (BoW)

A simple way to represent text as numerical vectors by counting word occurrences and representing each sentence as a vector or word counts

```
1 import pandas as pd
2 from sklearn.feature_extraction.text import CountVectorizer
3
4 corpus = [
5     "Large language models revolutionize business.",
6     "Business applications benefit from AI.",
7     "Language models learn from text data."
8 ]
9
10 # CountVectorizer is used to convert a collection of text documents to a matrix of token counts.
11 # The output is a sparse matrix where each row represents a document and each column represents a word in the corpus.
12 vectorizer = CountVectorizer()
13 X = vectorizer.fit_transform(corpus)
14
15 # Get feature names and the array representation of the matrix
16 feature_names = vectorizer.get_feature_names_out()
17
18 df = pd.DataFrame(X.toarray(), columns=feature_names)
19 df
```

	ai	applications	benefit	business	data	from	language	large	learn	models	revolutionize	text
0	0	0	0	1	0	0	1	1	0	1	1	0
1	1	1	1	1	0	1	0	0	0	0	0	0
2	0	0	0	0	1	1	1	0	1	1	0	1

## ✓ Term Frequency-Inverse Document Frequency (TF-IDF)

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2
3 corpus = [
4     "Large language models revolutionize business.",
5     "Business applications benefit from AI.",
6     "Language models learn from text data."
7 ]
8
9 tfidf_vectorizer = TfidfVectorizer()
10
11 # Transform the corpus into a document-term matrix
12 # Each row represents a document in the corpus
13 # Each column represents a unique word in the corpus
14 X_tfidf = tfidf_vectorizer.fit_transform(corpus)
15
16
17 df = pd.DataFrame(X_tfidf.toarray(), columns=tfidf_vectorizer.get_feature_names_out())
18 df
```

	ai	applications	benefit	business	data	from	language	large	learn	models	revolutionize	text
0	0.000000	0.000000	0.000000	0.393511	0.000000	0.000000	0.393511	0.51742	0.000000	0.393511	0.51742	0.000000
1	0.490479	0.490479	0.490479	0.373022	0.000000	0.373022	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000	0.459548	0.349498	0.349498	0.000000	0.459548	0.349498	0.000000	0.459548

## ✓ Practical Example

## Sentiment Analysis of Movie Reviews

This application will go through the entire text preprocessing pipeline and show how it contributes to a real-world NLP task. Setup

Dataset: Use a small dataset of movie reviews (positive and negative) - you could use a subset of IMDB reviews or create 10-15 simple examples. Visual Flow: Create a slide that shows the entire pipeline:

### ✓ Text Processing Pipeline

**Raw Text → Tokenization → Normalization → Stop Words Removal → Stemming/Lemmatization → BoW → Classification**

Next, we will do the following:

- Import the needed libraries
- Download any additional modules that are required
- Prepare the dataset i.e Text Corpus

```
1 import pandas as pd
2 import re
3 from nltk.tokenize import word_tokenize
4 from nltk.corpus import stopwords
5 from nltk.stem import PorterStemmer, WordNetLemmatizer
6 from sklearn.feature_extraction.text import CountVectorizer
7 from sklearn.model_selection import train_test_split
8 from sklearn.naive_bayes import MultinomialNB
9 from sklearn.metrics import accuracy_score, classification_report
10
11 # Sample movie reviews dataset
12 reviews = [
13     {"text": "This movie was absolutely fantastic! Great acting and storyline.", "sentiment": 1},
14     {"text": "I loved this film. The characters were so well developed.", "sentiment": 1},
15     {"text": "Amazing cinematography and directing. One of the best films I've seen.", "sentiment": 1},
16     {"text": "The acting was good but the story was too predictable.", "sentiment": 0},
17     {"text": "Terrible movie. I wasted two hours of my life.", "sentiment": 0},
18     {"text": "The special effects were amazing but everything else was boring.", "sentiment": 0},
19     {"text": "I enjoyed the action sequences but the dialogue was poorly written.", "sentiment": 0},
20     {"text": "Brilliant performance by the lead actor! Highly recommended.", "sentiment": 1},
21     {"text": "So disappointing. The trailer was better than the actual movie.", "sentiment": 0},
22     {"text": "A masterpiece of modern cinema. I was captivated throughout.", "sentiment": 1}
23 ]
24
25 # Create DataFrame
26 df = pd.DataFrame(reviews)
27 print("Original Data:")
28 print(df.head())
29 print("\n")
```

Original Data:

	text	sentiment
0	This movie was absolutely fantastic! Great act...	1
1	I loved this film. The characters were so well...	1
2	Amazing cinematography and directing. One of t...	1
3	The acting was good but the story was too pred...	0
4	Terrible movie. I wasted two hours of my life.	0

In the text segment below, we will define the required text processing functions:


1. tokenize text function
2. normalize\_tokens function
3. remove\_stopwords function
4. stem\_tokens function
5. lemmatize\_tokens function
6. preprocess\_text function, the pipeline that calls all the previous functions

```
1 # Step 1: Tokenization
2 def tokenize_text(text):
3     tokens = word_tokenize(text)
4     # print(f"Sentence after tokenization: {tokens}") # uncomment if need to see results
5     return tokens
```

```

6
7 # Step 2: Normalization
8 def normalize_tokens(tokens):
9     # Convert to lowercase and remove punctuation
10    normalized = [re.sub(r'[^\w\s]', '', token.lower()) for token in tokens]
11    normalized = [token for token in normalized if token] # Remove empty strings
12    # print(f"Tokens after normalization: {normalized}") # uncomment if need to see results
13    return normalized
14
15 # Step 3: Remove stop words
16 def remove_stopwords(tokens):
17     stop_words = set(stopwords.words('english'))
18     filtered = [token for token in tokens if token not in stop_words]
19     # print(f"Tokens after stopword removal: {filtered}") # uncomment if need to see results
20     return filtered
21
22 # Step 4a: Stemming
23 def stem_tokens(tokens):
24     stemmer = PorterStemmer()
25     stemmed = [stemmer.stem(token) for token in tokens]
26     # print(f"Tokens after stemming: {stemmed}") # uncomment if need to see results
27     return stemmed
28
29 # Step 4b: Lemmatization
30 def lemmatize_tokens(tokens):
31     lemmatizer = WordNetLemmatizer()
32     lemmatized = [lemmatizer.lemmatize(token) for token in tokens]
33     # print(f"Tokens after lemmatization: {lemmatized}") # uncomment if need to see results
34     return lemmatized
35
36 # Complete preprocessing pipeline
37 def preprocess_text(text, use_stemming=True):
38     tokens = tokenize_text(text)
39     normalized_tokens = normalize_tokens(tokens)
40     no_stopwords = remove_stopwords(normalized_tokens)
41
42     if use_stemming:
43         return stem_tokens(no_stopwords)
44     else:
45         return lemmatize_tokens(no_stopwords)
46
47 # Demonstrate the preprocessing pipeline on one example
48 print("PREPROCESSING PIPELINE DEMONSTRATION:")
49 sample_text = df['text'][0]
50 print(f"Original text: '{sample_text}'")
51 processed_tokens = preprocess_text(sample_text, use_stemming=True)
52 print(f"Processed tokens: {processed_tokens}")
53 print("\n")

```

 PREPROCESSING PIPELINE DEMONSTRATION:  
 Original text: 'This movie was absolutely fantastic! Great acting and storyline.'  
 Processed tokens: ['movi', 'absolut', 'fantast', 'great', 'act', 'storylin']

Apply the text processing pipeline on all our sentences (entire text corpus)

```

1 df['processed_text'] = df['text'].apply(lambda x: ' '.join(preprocess_text(x, use_stemming=True)))
2 df

```



	text	sentiment	processed_text
0	This movie was absolutely fantastic! Great act...	1	movi absolut fantast great act storylin
1	I loved this film. The characters were so well...	1	love film charact well develop
2	Amazing cinematography and directing. One of t...	1	amaz cinematographi direct one best film seen
3	The acting was good but the story was too pred...	0	act good stori predict
4	Terrible movie. I wasted two hours of my life.	0	terribl movi wast two hour life
5	The special effects were amazing but everythin...	0	special effect amaz everyth els bore
6	I enjoyed the action sequences but the dialogu...	0	enjoy action sequenc dialogu poorli written
7	Brilliant performance by the lead actor! Highl...	1	brilliant perform lead actor highli recommend
8	So disappointing. The trailer was better than ...	0	disappoint trailer better actual movi
9	A masterpiece of modern cinema. I was captivat...	1	masterpiec modern cinema captiv throughout

Convert text into numerical representation using the Bag of Words Method, we will the use the simple count approach

```
1 # Bag of Words representation
2 vectorizer = CountVectorizer()
3 X = vectorizer.fit_transform(df['processed_text'])
4
5 print("BoW Matrix (First 3 Documents):")
6 bow_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())
7 bow_df
```



BoW Matrix (First 3 Documents):

	absolut	act	action	actor	actual	amaz	best	better	bore	brilliant	...	special	stori	storylin	terribl	throughout
0	1	1	0	0	0	0	0	0	0	0	...	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	0	...	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	...	0	1	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0
5	0	0	0	0	0	1	0	0	1	0	...	1	0	0	0	0
6	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	1	...	0	0	0	0	0
8	0	0	0	0	1	0	0	1	0	0	...	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1

10 rows × 51 columns

Create the classifier and train it on the generated BoW representations

```
1 X_train, X_test, y_train, y_test = train_test_split(X, df['sentiment'], test_size=0.3, random_state=42)
2
3 # Train a Naive Bayes classifier
4 clf = MultinomialNB()
5 clf.fit(X_train, y_train)
```



▼ MultinomialNB ⓘ ?  
MultinomialNB()

Use the trained classifier to make predictions using the test data (sentences that was not seen by the classifier during training)

```
1 y_pred = clf.predict(X_test)
```

Evaluate the classification results (Model Performance)

```

1 print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
2 print("\nClassification Report:")
3 print(classification_report(y_test, y_pred))

```

→ Accuracy: 0.67

```

Classification Report:
              precision    recall  f1-score   support

     0           1.00        0.50        0.67         2
     1           0.50        1.00        0.67         1

 accuracy          0.67         0.67         0.67         3
 macro avg          0.75         0.75         0.67         3
 weighted avg       0.83         0.67         0.67         3

```

---

## ✓ Part 2:- Language Models

### ✓ Building a Simple N-gram Language Model

Import the required libraries

```

1 import random
2 from collections import defaultdict

```

Define the function that builds the language model

```

1 def build_ngram_model(text, n=2):
2     """Build an n-gram language model from text."""
3     tokens = word_tokenize(text.lower())
4     ngrams_dict = defaultdict(list)
5
6     # Create dictionary of n-grams and possible next words
7     for i in range(len(tokens) - n):
8         current_ngram = tuple(tokens[i:i+n])
9         next_word = tokens[i+n]
10        ngrams_dict[current_ngram].append(next_word)
11
12    return ngrams_dict

```

Define the function that is used to generate new text based on an input seed text

```

1 def generate_text(model, seed, length):
2     """Generate text using the n-gram model."""
3     current = seed
4     result = list(seed)
5
6     for _ in range(length):
7         if current in model:
8             # Randomly select a possible next word
9             next_word = random.choice(model[current])
10            result.append(next_word)
11            # Update current n-gram
12            current = current[1:] + (next_word,)
13        else:
14            # If current n-gram is not in model, break
15            break
16
17    return ' '.join(result)

```

Create a sample corpus to build our language model

```

1 # Sample text corpus
2 corpus = """Large language models are transforming how businesses operate.
3 These models can understand language, generate text, and perform various tasks.

```

```

4 Businesses use language models for customer service, content creation, and data analysis.
5 Language models learn patterns from vast amounts of text data."""
6
7 # Build a bigram model
8 bigram_model = build_ngram_model(corpus, 2)
9
10 print("Bigram Dictionary/Model:")
11 for key, value in bigram_model.items():
12     print(f"{key} → {value}")

```

```

↔ Bigram Dictionary/Model:
('large', 'language') → ['models']
('language', 'models') → ['are', 'for', 'learn']
('models', 'are') → ['transforming']
('are', 'transforming') → ['how']
('transforming', 'how') → ['businesses']
('how', 'businesses') → ['operate']
('businesses', 'operate') → ['.']
('operate', '.') → ['these']
('.', 'these') → ['models']
('these', 'models') → ['can']
('models', 'can') → ['understand']
('can', 'understand') → ['language']
('understand', 'language') → [',']
('language', ',') → ['generate']
(',', 'generate') → ['text']
('generate', 'text') → [',']
('text', ',') → ['and']
(',', 'and') → ['perform', 'data']
('and', 'perform') → ['various']
('perform', 'various') → ['tasks']
('various', 'tasks') → ['.']
('tasks', '.') → ['businesses']
('.', 'businesses') → ['use']
('businesses', 'use') → ['language']
('use', 'language') → ['models']
('models', 'for') → ['customer']
('for', 'customer') → ['service']
('customer', 'service') → [',']
('service', ',') → ['content']
(',', 'content') → ['creation']
('content', 'creation') → [',']
('creation', ',') → ['and']
('and', 'data') → ['analysis']
('data', 'analysis') → ['.']
('analysis', '.') → ['language']
('.', 'language') → ['models']
('models', 'learn') → ['patterns']
('learn', 'patterns') → ['from']
('patterns', 'from') → ['vast']
('from', 'vast') → ['amounts']
('vast', 'amounts') → ['of']
('amounts', 'of') → ['text']
('of', 'text') → ['data']
('text', 'data') → ['.']

```

Use the language model we built to predict new words base on an input text

Make sure to run the code below several times to see how the language model generates new text everytime (**A Non-Deterministic Stochastic Process**)

```

1 # Generate text using the model
2 seed = ('language', 'models')
3 generated_text = generate_text(bigram_model, seed, 20)
4 print(generated_text)

```


↔ language models are transforming how businesses operate . these models can understand language , generate text , and data an

## ✓ Evaluating Language Models: Perplexity

Perplexity measures how well a language model predicts a sample:



```
1 import numpy as np
2 from collections import Counter
3
4 def get_bigram_probability(model, unigram_counts, vocab_size, word1, word2):
5     """Calculate the probability of a bigram."""
6     bigram = (word1, word2)
7     bigram_count = len(model[bigram])
8     unigram_count = unigram_counts[word1]
9     # Add-one smoothing
10    probability = (bigram_count + 1) / (unigram_count + vocab_size)
11    return probability
12
13 def calculate_perplexity(test_text, model, unigram_counts, vocab_size):
14     """Calculate perplexity of test text using the bigram model."""
15     tokens = word_tokenize(test_text.lower())
16     log_probability = 0
17
18     for i in range(len(tokens) - 1):
19         bigram = (tokens[i], tokens[i+1])
20         probability = get_bigram_probability(model, unigram_counts, vocab_size, tokens[i], tokens[i+1])
21         log_probability += np.log2(probability)
22
23     # Perplexity = 2^(-average log probability)
24     perplexity = 2 ** (-log_probability / (len(tokens) - 1))
25     return perplexity
26
27 # Calculate unigram counts and vocabulary size
28 tokens = word_tokenize(corpus.lower())
29 unigram_counts = Counter(tokens)
30 vocab_size = len(unigram_counts)
31
32 # Test the model on new text
33 test_text = "Language models help businesses understand customer feedback."
34 perplexity = calculate_perplexity(test_text, bigram_model, unigram_counts, vocab_size)
35 print(f"Perplexity: {perplexity:.2f}")
```

 Perplexity: 28.45

---