**BICOL UNIVERSITY COLLEGE OF SCIENCE**
CS Elective – Artificial Intelligence
Coding Exercises - 3

```
# "Lastname_Firstname_Pre-Processing.ipynb"
# ==============================================
# INSTALL DEPENDENCIES
# ==============================================
!pip install nltk spacy --quiet
!python -m spacy download en_core_web_sm

import re
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import spacy
```

Purpose:

- Stopwords = common words (a, the, is...) normally removed because they add no meaning.
- Lemmatizer = reduces words to dictionary form (better for meaning).
- Stemmer = cuts off word endings (faster but less accurate).

```
# Load spaCy model
nlp = spacy.load("en_core_web_sm")

# ==============================================
# SAMPLE DATASET
# ==============================================
texts = [
    "Natural Language Processing (NLP) is AMAZING!!!",
    "Python provides powerful tools for text cleaning & tokenization.",
    "Students should complete the pre-processing tasks on time."
]

print("Original Texts:")
for t in texts:
    print("-", t)
```

```
# ==============================================
# PREPROCESSING FUNCTIONS
# ==============================================

stop_words = set(stopwords.words("english"))
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()

def clean_text(text):
    text = re.sub(r"[^a-zA-Z\s]", "", text)
    return text
```

**What this does:**

- Removes everything **not a letter or whitespace**
  → punctuation, numbers, symbols, emojis.

Example:
"Hello!!! I'm 21 years old :)" → "Hello Im years old"

**Why important:**
This simplifies text for models that expect alphabetic tokens only.

```
def preprocess(text):
    print("\n==========================================")
    print("Original:", text)

    # ---------- 1. DATA CLEANING ----------
    cleaned = clean_text(text)
    print("Cleaned:", cleaned)
    #    Removes punctuation, digits, emojis, etc.

    # ---------- 2. TOKENIZATION ----------
    tokens = word_tokenize(cleaned)
    print("Tokens:", tokens)
    # Splits the sentence into individual words.

    # ---------- 3. LOWERCASE ----------
    lower_tokens = [t.lower() for t in tokens]
    print("Lowercased:", lower_tokens)
    # Models treat Cat ≠ cat, so everything must be consistent.

    # ---------- 4. STOP WORDS REMOVAL ----------
    no_stop = [t for t in lower_tokens if t not in stop_words]
    print("No Stop Words:", no_stop)
```

- ✓ Removes meaningless words like: the, is, am, are, to, of, in, that, this, etc.

✓ This reduces noise and improves algorithm performance.

```
# ---------- 5. LEMMATIZATION ----------
lemmas = [lemmatizer.lemmatize(t) for t in no_stop]
print("Lemmatized:", lemmas)
```

Converts words to their dictionary/root form.
Examples:

- “cars” → “car”
- “running” → “run”
- “better” → “good”
- ✓ Improves generalization for ML models.

```
# ---------- 6. STEMMING ----------
stems = [stemmer.stem(t) for t in no_stop]
print("Stemmed:", stems)
```

Cuts words to their root form:

- “playing” → “play”
- “studies” → “studi”
- “better” → “better” (unchanged)
- ✓ Good for search engines.
- ✓ NOT always good for ML because it can distort meaning.

```
# ---------- 7. POS TAGGING ----------
doc = nlp(" ".join(tokens))
pos_tags = [(token.text, token.pos_) for token in doc]
print("POS Tags:", pos_tags)
```

Uses spaCy to get:

- Noun
- Verb
- Adjective
- Adverb
- Proper noun
- Determiner
- Pronoun
- … etc.

```
# ==========================================
# APPLY TO ALL TEXTS
# ==========================================
for t in texts:
    preprocess(t)
# Feature Extraction
```

```
# Lastname_Firstname_Feature_Extraction.ipynb
# ==========================================================
# INSTALL REQUIRED LIBRARIES
# ==========================================================
!pip install nltk gensim scikit-learn --quiet
```

- **nltk** → tokenization
- **gensim** → Word2Vec
- **scikit-learn** → BoW, TF-IDF
- **pandas** → convert matrices to DataFrames

```
# IMPORTS
import nltk
nltk.download('punkt')

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import pandas as pd

# ==========================================================
# SAMPLE DATASET
# ==========================================================
texts = [
   "Natural Language Processing enables computers to understand text.",
   "Machine learning provides tools for analyzing large volumes of data.",
   "Deep learning models require a lot of high quality training data."
]

# ==========================================================
# BAG OF WORDS
# ==========================================================
vectorizer = CountVectorizer()
bow_matrix = vectorizer.fit_transform(texts)

bow_df = pd.DataFrame(bow_matrix.toarray(),
columns=vectorizer.get_feature_names_out())
print("=== B A G   O F   W O R D S ===")
display(bow_df)
```

- ✓ Builds a vocabulary of all unique words.
- ✓ Counts how many times each word appears in every sentence.

**Good for:**

- Traditional ML algorithms (SVM, LR, Naive Bayes)
- Simple text classification

**Limitations:**

- Ignores grammar & order
- High-dimensional (many columns)

```
# =========================================================
# N-GRAMS (Unigrams, Bigrams, Trigrams)
# =========================================================
ngram_vectorizer = CountVectorizer(ngram_range=(1,3))
ngram_matrix = ngram_vectorizer.fit_transform(texts)

ngram_df = pd.DataFrame(ngram_matrix.toarray(),
columns=ngram_vectorizer.get_feature_names_out())
print("\n=== N - G R A M S (1 to 3) ===")
display(ngram_df)
```

Creates features for:

- **Unigrams** → 1 word: "learning"
- **Bigrams** → 2 words: "machine learning"
- **Trigrams** → 3 words: "high quality training"

✓ Captures **context** and **meaning** better than BoW.

Example:

- Method -> "not good" meaning
- Unigram -> "not" + "good" (can't detect negativity)
- Bigram -> "not good" (captures negative sentiment)

**Use cases:**

- Sentiment analysis
- Spam detection
- Intent classification

```
# =========================================================
# TF–IDF
# =========================================================
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(texts)

tfidf_df = pd.DataFrame(tfidf_matrix.toarray(),
columns=tfidf_vectorizer.get_feature_names_out())
print("\n=== T F - I D F ===")
display(tfidf_df)
```

Term Frequency × Inverse Document Frequency

- Words that appear many times in one document → high weight

- Words that appear in all documents → low weight (not useful)

TF-IDF is better than BoW:

- Reduces weight of common words like “data,” “learning”
- Highlights important keywords

Example weights (illustrative):

| Term | Text 1 | Text 2 | Text 3 |
|---|---|---|---|
| **processing** | 0.7 | 0.0 | 0.0 |
| **learning** | 0.0 | 0.6 | 0.6 |
| **data** | 0.0 | 0.3 | 0.4 |

```
# =======================================================
# WORD2VEC
# =======================================================
tokenized_texts = [word_tokenize(text.lower()) for text in texts]

w2v_model = Word2Vec(sentences=tokenized_texts, vector_size=50, window=5,
min_count=1, workers=4)

print("\n=== W O R D 2 V E C : Vocabulary ===")
print(list(w2v_model.wv.index_to_key))

print("\n=== Vector for word 'data' ===")
print(w2v_model.wv["data"])

print("\n=== Most similar to 'data' ===")
print(w2v_model.wv.most_similar("data"))
```

Word2Vec does:

- Learns semantic meaning of words by looking at context.
- Instead of counts, each word becomes a dense numeric vector (e.g., 50 dimensions).
- Meaning:
  - “data” appears in similar contexts as “training,” “volumes,” etc.

**Strengths:**

- Captures meaning
- Maintains relations (king - man + woman = queen)
- Useful for deep learning

**Weaknesses:**

- Needs many sentences to train well
- Models trained on small corpus may be weak