Week 3 – PRACTICAL

Preprocessing, POS Tagging and Feature Extraction (BoW & TF-IDF)

Introduction

This practical builds on the preprocessing and text-analytics concepts introduced in the lecture.

- You will clean and prepare a real dataset, perform part-of-speech (POS) tagging, and extract text features using both Bag-of-Words (BoW) and TF-IDF representations.
- Finally, you will visualise the resulting vocabulary and reflect on ethical and fairness issues in text analysis.

Tools Overview

- **NLTK (Natural Language Toolkit)** – tokenisation, stopword lists, lemmatisation, POS tagging
- **spaCy** – efficient POS tagging and syntactic features
- **scikit-learn** – `CountVectorizer`, `TfidfVectorizer` for feature extraction
- **matplotlib / wordcloud** – simple text visualisations
- **Dataset**: `spam.csv` (SMS Spam Collection Dataset, provided in NILE)

Part A – Core Tasks (Complete in Class)

Q1). Load and Inspect the Dataset

```
import pandas as pd
df = pd.read_csv("spam.csv", encoding='latin-1')[['v1', 'v2']]
df.columns = ['label', 'message']
print(df.head())
print(df['label'].value_counts())
```

- Reflection – Is the dataset balanced?
- How might this imbalance influence model fairness later?

Q2). Preprocess the Text

Clean and normalise messages by lower-casing, removing punctuation and stopwords, and applying lemmatisation.

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

nltk.download('punkt'); nltk.download('wordnet'); nltk.download('stopwords')
stops = set(stopwords.words('english'))
lemma = WordNetLemmatizer()

def preprocess(text):
    tokens = word_tokenize(text.lower())
    words = [w for w in tokens if w.isalpha() and w not in stops]
    return " ".join(lemma.lemmatize(w) for w in words)

df['clean'] = df['message'].apply(preprocess)
print(df[['message','clean']].head())
```

- Reflection - Would you keep numbers if messages contained discount codes? Why?

Q3). POS Tagging

Tag each token with its grammatical category.

(a) Try using NLTK

```
import nltk
tokens = nltk.word_tokenize("Free tickets available for winners now!")
print(nltk.pos_tag(tokens))
```

(b) Try using spaCy

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Artificial intelligence is transforming industries.")
for t in doc:
    print(t.text, t.pos_)
```

Observe the Part of speech tagging in NLTK and spaCy.

(c) Now, for your cleaned Spam dataset, extract only NOUNS and ADJECTIVES from `df['clean']` and store them in a new column `filtered_text`.

- Hint: Keep tags `NN`, `NNS`, `JJ`, `JJR`, `JJS`.

```
from nltk import word_tokenize, pos_tag

def extract_nouns_adjectives(text):
    # Tokenize the text
    tokens = word_tokenize(text)
    # POS tag the tokens
    tagged = pos_tag(tokens)
    # Keep only nouns and adjectives
    filtered = [word for word, tag in tagged if tag in ['NN', 'NNS', 'JJ', 'JJR', 'JJS']]
    # Return them joined as a clean string
    return " ".join(filtered)

df['filtered_text'] = df['clean'].apply(extract_nouns_adjectives)

print(df[['label', 'message', 'filtered_text']].head())
```

- Question – Why might you choose to focus on nouns and adjectives for feature extraction?

Q4). Bag-of-Words Representation

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(ngram_range=(1,2))
X_bow = vectorizer.fit_transform(df['clean'])
print("Shape:", X_bow.shape)
```

- Reflection – Inspect the first 20 features.
- Which bigrams look spam-like or promotional?

```
print(vectorizer.get_feature_names_out()[:20])
```

- Inspect more to have a better view

Q5). TF-IDF Representation

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(ngram_range=(1,2))
X_tfidf = tfidf.fit_transform(df['clean'])
print("Shape:", X_tfidf.shape)
```

$$TF-IDF(t,d) = TF(t,d) \times \log\left(\frac{N}{DF(t)}\right)$$

where TF(t,d) = term frequency of t in d,

DF(t) = number of documents containing t,

N = total documents,

log(N / DF(t)) = inverse document frequency.

- Think – Which terms get the highest weights? Why might that be important?
- Observe the top terms

```
import numpy as np
feature_names = vectorizer.get_feature_names_out()
mean_tfidf = np.asarray(X_tfidf.mean(axis=0)).ravel()
top_ids = mean_tfidf.argsort()[::-1][:20]
top_terms = feature_names[top_ids]
print(top_terms)
```

Part B – Extended Tasks

Q6). POS-Filtered TF-IDF (Compare Effect)

Apply TF-IDF on the POS-filtered text created earlier (`filtered_text`).

```
X_filtered = tfidf.fit_transform(df['filtered_text'])
print("Original:", X_tfidf.shape, "Filtered:", X_filtered.shape)
```

Question

- How does limiting features to nouns/adjectives change the number of dimensions?
- Does it make the text representation more focused or less informative?

Q7). Exploratory Text Analytics

(a) Word Frequency Plot

```
from collections import Counter
import matplotlib.pyplot as plt

all_words = " ".join(df['clean']).split()
common = Counter(all_words).most_common(20)
words, counts = zip(*common)

plt.barh(words, counts)
plt.title("Top 20 Frequent Words")
plt.gca().invert_yaxis()
plt.xlabel("Frequency")
plt.show()
```

(b) Word Cloud

```
from wordcloud import WordCloud
spam_text = " ".join(df[df.label=="spam"]["clean"])
ham_text  = " ".join(df[df.label=="ham"]["clean"])
WordCloud(width=800,height=400).generate(spam_text).to_image()
WordCloud(width=800,height=400).generate(ham_text).to_image()
```

For a more detailed wordcloud that separates the dataset:

```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

spam_text = " ".join(df[df.label == "spam"]["clean"])
ham_text  = " ".join(df[df.label == "ham"]["clean"])

# Create word clouds
spam_wc = WordCloud(width=800, height=400, background_color='white').generate(spam_text)
ham_wc  = WordCloud(width=800, height=400, background_color='white').generate(ham_text)

# Plot side by side
plt.figure(figsize=(14,6))
plt.subplot(1,2,1)
plt.imshow(spam_wc, interpolation='bilinear')
plt.title("Spam Messages Word Cloud")
plt.axis('off')

plt.subplot(1,2,2)
plt.imshow(ham_wc, interpolation='bilinear')
plt.title("Ham Messages Word Cloud")
plt.axis('off')

plt.show()
```

- Reflection – Which words dominate in spam messages? What do they reveal about tone or intent?

Q8). Fairness and Bias Reflection

Discuss briefly:

a) How could class imbalance result in bias in a spam-filtering model?
b) What ethical concerns arise when cleaning text that contains dialect or slang?
c) How can we document preprocessing choices to remain transparent?

Q9). Extended Reflection Questions

a) Does POS tagging help in selecting more meaningful features? Why?
b) How do BoW and TF-IDF differ in handling common vs. rare words?
c) Which preprocessing decisions do you think would influence model accuracy most?
d) What potential biases can arise from heavy text cleaning?
e) How does this practical connect to the NLP pipeline discussed in the lecture?