Beni-Suef University
College of Computers and AI
Department of Computer Science

Lab Manual
Knowledge Base Systems

Preparing the scientific material

Dr. Noha Yehia

T.A. Ehab Ibrahim

Course	Knowledge Base Systems — NLP
Sections	10 Sections covering Introduction to Transformers & GUI
Dataset	amazon_alexa.tsv (used throughout all sections)
Language	Python 3.x
Authors	Dr. Noha Yehia and T.A Ehab Ibrahim

Course Sections Overview

Section 1: Introduction to NLP

Section 2: Feature Engineering

Section 3: Data Cleaning

Section 4: Feature Extraction — BoW, TF-IDF, N-Grams

Section 5: Data Visualization

Section 6: Text Classification — Naive Bayes & SVM

Section 7: Transformers & BERT (Extended)

Section 8: Large Language Models — LLMs (Extended)

Section 9: Saving & Loading Trained Models (Extended)

Section 10: Python GUI for NLP Apps (Extended)

1 Introduction to NLP

Objective	Understand what NLP is, why it matters, its applications, and key libraries
Dataset	amazon_alexa.tsv
Libraries	pandas, textblob, nltk, spacy

1.1 What is NLP?

Natural Language Processing (NLP) is a branch of Artificial Intelligence that enables computers to understand, interpret, manipulate, and generate human language in a way that is both meaningful and useful. It sits at the intersection of three fields:

- Linguistics — Grammar, syntax, semantics, and pragmatics
- Computer Science — Algorithms, data structures, and machine learning
- Artificial Intelligence — Learning, reasoning, and perception

> 🔑 **Key Definition**
>
> NLP bridges the gap between human communication and machine understanding. Every time you use Google Search, Alexa, or Gmail's spam filter — NLP is working behind the scenes.

1.2 Why Should You Learn NLP?

Metric	Value	What it means
NLP Market Size	$43 Billion by 2025	Massive commercial investment
Unstructured Data	80% of business data is text	Most data is untapped
Engineer Demand	3× growth rate	High-paying career field

1.3 Applications of NLP

- Machine Translation — Google Translate, DeepL
- Sentiment Analysis — Analyzing reviews, tweets, and feedback
- Chatbots & Virtual Assistants — Siri, Alexa, ChatGPT

- Text Summarization — Condensing long documents automatically
- Named Entity Recognition — Extracting people, places, organizations
- Spam Detection — Filtering emails automatically
- Information Retrieval — Search engines understanding queries

1.4 Steps to Solve NLP Problems

1. Problem Definition — Understand the task: classification, summarization, translation
2. Data Collection — Gather raw text data from datasets, APIs, or web scraping
3. Data Cleaning — Remove noise, handle punctuation, stopwords, and encoding issues
4. Feature Engineering — Extract useful features: word counts, polarity, POS tags
5. Feature Extraction — Convert text to numeric form: BoW, TF-IDF, embeddings
6. Model Building — Select and train ML/DL models on the prepared feature matrix
7. Evaluation — Measure accuracy, F1, precision, recall on test data

1.5 Lab — Loading and Exploring the Dataset

Install Required Libraries

```
# Run this cell once to install all required libraries
!pip install textblob wordcloud nltk seaborn scikit-learn


import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('stopwords')
nltk.download('wordnet')
print('All libraries ready!')
```

Load and explore amazon_alexa.tsv

```
import pandas as pd

# Load the dataset — note sep='\t' for tab-separated files
df = pd.read_csv('amazon_alexa.tsv', sep='\t', encoding='utf-8')

# Basic exploration
print('Shape:', df.shape)             # (rows, columns)
print('Columns:', df.columns.tolist())
print()
print(df.head())                      # first 5 rows
print()
print(df.info())                      # data types and nulls
print()
print(df.describe())                  # statistical summary
print()
print('Missing values:',)
print(df.isnull().sum())
print()
print('Feedback distribution:')
print(df['feedback'].value_counts())
```

Popular NLP Libraries

Library	Best For	Key Features
Textblob	Sentiment analysis, quick tasks	Polarity, subjectivity, spelling
NLTK	Learning & research	Tokenization, POS tagging, stemming
spaCy	Production NLP	NER, dependency parsing, fast
Gensim	Topic modeling	Word2Vec, Doc2Vec, LDA
Scikit-learn	ML classifiers	TF-IDF, Naive Bayes, SVM
Transformers	Deep learning NLP	BERT, GPT, fine-tuning

2 Feature Engineering

Objective	Extract numerical features from text: length, polarity, POS counts
Dataset	amazon_alexa.tsv
Libraries	pandas, textblob, nltk, seaborn, matplotlib

2.1 What is Feature Engineering?

Feature Engineering transforms raw text into meaningful numerical signals that machine learning algorithms can use. Since ML models cannot read raw text, we create numeric columns that capture what the text is about.

Feature Type	Examples	What it captures
Surface	Character count, word count, sentence count	Length and structure
Lexical	Unique word ratio, stop word count	Vocabulary richness
Syntactic	Noun count, verb count, adjective count	Grammar patterns
Sentiment	Polarity, subjectivity, emotion intensity	Opinion signals

2.2 Text Length Features

```
import pandas as pd
import nltk

df = pd.read_csv('amazon_alexa.tsv', sep='\t', encoding='utf-8')
df['text'] = df['verified_reviews'].astype(str)

# Character count — total characters including spaces
df['char_count'] = df['text'].apply(len)

# Word count — number of words
df['word_count'] = df['text'].apply(lambda x: len(x.split()))
```

```
# Sentence count — number of sentences
df['sent_count'] = df['text'].apply(
    lambda x: len(nltk.sent_tokenize(x)))

# Average word length
df['avg_word_len'] = df['text'].apply(
    lambda x: sum(len(w) for w in x.split()) / max(len(x.split()),1))

print(df[['char_count','word_count','sent_count','avg_word_len']].describe())
```

2.3 Polarity & Subjectivity

TextBlob computes two sentiment scores for each text: Polarity (−1 to +1, how positive/negative) and Subjectivity (0 to 1, how opinion-based vs factual).

```
from textblob import TextBlob

# Polarity: -1 (very negative) to +1 (very positive)
df['polarity'] = df['text'].apply(
    lambda x: TextBlob(x).sentiment.polarity)

# Subjectivity: 0 (fully objective) to 1 (fully subjective)
df['subjectivity'] = df['text'].apply(
    lambda x: TextBlob(x).sentiment.subjectivity)

print(df[['polarity','subjectivity']].describe())
print()
print('Sample:',)
print(df[['text','polarity','subjectivity']].head(3))
```

2.4 POS Tag Features

```
import string

# Punctuation count
df['punct_count'] = df['text'].apply(
    lambda x: sum(1 for c in x if c in string.punctuation))

# POS tag feature extraction
def count_pos(text, target_tag):
    """Count tokens with a specific POS tag prefix."""
    try:
        tokens = nltk.word_tokenize(text)
        tagged = nltk.pos_tag(tokens)
        return sum(1 for _, tag in tagged
                   if tag.startswith(target_tag))
    except: return 0

# Sample 500 rows for speed
sample = df.sample(500, random_state=42)
sample['noun_count'] = sample['text'].apply(
    lambda x: count_pos(x,'NN'))
sample['verb_count'] = sample['text'].apply(
    lambda x: count_pos(x,'VB'))
sample['adj_count']  = sample['text'].apply(
    lambda x: count_pos(x,'JJ'))
sample['adv_count']  = sample['text'].apply(
    lambda x: count_pos(x,'RB'))

print(sample[['noun_count','verb_count',
              'adj_count','adv_count']].describe())
```

3 Data Cleaning

Objective	Clean and normalize raw text for ML pipelines
Dataset	amazon_alexa.tsv
Libraries	re, string, unicodedata, nltk

3.1 Why Data Cleaning Matters

"80% of data science work is data cleaning. Raw text is messy, inconsistent, and full of noise that confuses ML models."

Problem	Example	Solution
Punctuation	'Hello!' vs 'Hello'	Remove or standardize
HTML tags	text	Strip with regex
URLs	http://buy.it/now	Remove or replace
Accented chars	café → cafe	Unicode normalization
Case mismatch	NLP vs nlp vs N.L.P.	Lowercase everything
Stop words	the, is, and, of	Remove from NLTK list

3.2 Removing Punctuation & Numbers

```
import re, string

text = 'Hello, World! 123 #NLP is Amazing!!!'

# Method 1: Remove punctuation using str.translate()
no_punct = text.translate(
    str.maketrans("", string.punctuation))
print('No punct:', no_punct)

# Method 2: Remove numbers with regex
no_nums = re.sub(r'\d+', '', text)
print('No nums:', no_nums)
```

```
# Method 3: Keep only letters and spaces
letters_only = re.sub(r'[^a-zA-Z\s]', '', text)
print('Letters only:', letters_only)
```

3.3 Tokenization

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize

text = 'Data cleaning is crucial! Let us start now.'

# Word tokenization
word_tokens = word_tokenize(text)
print('Words:', word_tokens)
# ['Data','cleaning','is','crucial','!','Let','us','start','now','.']

# Sentence tokenization
sent_tokens = sent_tokenize(text)
print('Sentences:', sent_tokens)
# ['Data cleaning is crucial!', 'Let us start now.']

# Apply to dataset
df['tokens'] = df['text'].apply(word_tokenize)
print(df[['text','tokens']].head(3))
```

3.4 Stop Words Removal

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

# Load English stop words (179 words)
stop_words = set(stopwords.words('english'))
print('Total stop words:', len(stop_words))
print('Sample:', list(stop_words)[:10])

text = 'This is a great product and I love it'
tokens = word_tokenize(text.lower())
clean  = [w for w in tokens if w not in stop_words]
```

```
print('Original:', tokens)
print('Cleaned: ', clean)
# Cleaned: ['great', 'product', 'love']
```

3.5 Stemming & Lemmatization

```
from nltk.stem import PorterStemmer, WordNetLemmatizer

ps  = PorterStemmer()
wnl = WordNetLemmatizer()

words = ['running','studies','better','was','geese','caring']

print(f'{'Word':15} {'Stem':15} {'Lemma':15}')
print('-' * 45)
for word in words:
    stem  = ps.stem(word)
    lemma = wnl.lemmatize(word, pos='v')
    print(f'{word:15} {stem:15} {lemma:15}')
```

3.6 Complete Cleaning Pipeline

```
import re, string, unicodedata, nltk
from nltk.tokenize import word_tokenize
from nltk.corpus   import stopwords
from nltk.stem     import WordNetLemmatizer

stop_words = set(stopwords.words('english'))
wnl        = WordNetLemmatizer()

def clean_text(text):
    text = str(text).lower()                          # 1. Lowercase
    text = unicodedata.normalize('NFKD', text)        # 2. Normalize unicode
    text = text.encode('ascii','ignore').decode()     # 3. Remove accented chars
    text = re.sub(r'<.*?>', '', text)                 # 4. Strip HTML tags
    text = re.sub(r'http\S+', '', text)               # 5. Remove URLs
    text = re.sub(r'[^a-z\s]', '', text)              # 6. Keep letters only
    tokens = word_tokenize(text)                      # 7. Tokenize
    tokens = [wnl.lemmatize(w) for w in tokens        # 8. Lemmatize
              if w not in stop_words and len(w) > 1]  # 9. Remove stop words
    return ' '.join(tokens)
```

```
df['clean_text'] = df['text'].apply(clean_text)
print('Before:', df['text'].iloc[0])
print('After: ', df['clean_text'].iloc[0])
```

4 Feature Extraction — BoW, TF-IDF & N-Grams

Objective	Convert cleaned text to numeric vectors for ML models
Dataset	amazon_alexa.tsv (clean_text column)
Libraries	scikit-learn, nltk, collections

4.1 Bag of Words (BoW)

Bag of Words represents text as word frequency vectors. Each document becomes a vector where each element is the count of a vocabulary word. Grammar and word order are ignored — only counts matter.

```
from sklearn.feature_extraction.text import CountVectorizer

# Simple demonstration
corpus = [
    'I love Alexa',
    'I love Python',
    'Alexa is amazing',
]

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

print('Vocabulary:', vectorizer.vocabulary_)
print('BoW Matrix:')
print(X.toarray())

# Apply to amazon_alexa dataset
bow = CountVectorizer(max_features=5000)
X_bow = bow.fit_transform(df['clean_text'])
print('\nDataset BoW shape:', X_bow.shape)
print('Vocabulary size:', len(bow.vocabulary_))
```

4.2 TF-IDF

TF-IDF (Term Frequency — Inverse Document Frequency) weights words by how important they are to a specific document relative to the whole corpus. Common words score lower; distinctive words score higher.

Formula

TF-IDF(word, doc) = TF(word in doc) × log(N / docs containing word) TF = how often the word appears in this document IDF = how rare the word is across all documents

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Basic TF-IDF
tfidf = TfidfVectorizer(max_features=5000)
X_tfidf = tfidf.fit_transform(df['clean_text'])

print('TF-IDF Matrix shape:', X_tfidf.shape)

# Show top words by TF-IDF score
import numpy as np
feature_names = tfidf.get_feature_names_out()
scores = np.array(X_tfidf.mean(axis=0)).flatten()
top_idx = scores.argsort()[::-1][:15]
print('\nTop 15 words by TF-IDF score:')
for i in top_idx:
    print(f'  {feature_names[i]:20} {scores[i]:.4f}')

# Advanced TF-IDF with bigrams
tfidf_adv = TfidfVectorizer(
    max_features=10000,
    ngram_range=(1,2),     # include unigrams AND bigrams
    min_df=2,              # ignore words in < 2 docs
    max_df=0.95,           # ignore words in > 95% of docs
    sublinear_tf=True,     # apply log to TF scores
)
X_adv = tfidf_adv.fit_transform(df['clean_text'])
print('\nAdvanced TF-IDF shape:', X_adv.shape)
```

4.3 N-Gram Analysis

```
from collections import Counter
from nltk import ngrams


def get_top_ngrams(texts, n, top_k=15):
    all_ngrams = Counter()
    for text in texts:
        words = text.split()
        if len(words) >= n:
            all_ngrams.update(ngrams(words, n))
    return all_ngrams.most_common(top_k)


# Top bigrams
print('Top 10 Bigrams:')
for gram, count in get_top_ngrams(df['clean_text'], 2, 10):
    print(f'  {" ".join(gram):25} {count}')


# Top trigrams
print('\nTop 10 Trigrams:')
for gram, count in get_top_ngrams(df['clean_text'], 3, 10):
    print(f'  {" ".join(gram):35} {count}')
```

5 Data Visualization for NLP

Objective	Visualize text features to gain insights before modeling
Dataset	amazon_alexa.tsv
Libraries	matplotlib, seaborn, wordcloud, collections

5.1 Polarity & Subjectivity Plots

```
import matplotlib.pyplot as plt
from textblob import TextBlob

df['polarity']     = df['text'].apply(
    lambda x: TextBlob(str(x)).sentiment.polarity)
df['subjectivity'] = df['text'].apply(
    lambda x: TextBlob(str(x)).sentiment.subjectivity)

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Polarity histogram
axes[0].hist(df['polarity'], bins=40, color='teal', edgecolor='white')
axes[0].axvline(0, color='gray', linestyle='--', linewidth=1)
axes[0].set_title('Polarity Distribution')
axes[0].set_xlabel('Polarity Score (← Negative   Positive →)')

# Scatter: polarity vs subjectivity
colors = df['feedback'].map({1:'teal', 0:'crimson'})
axes[1].scatter(df['polarity'], df['subjectivity'],
                c=colors, alpha=0.3, s=15)
axes[1].set_title('Polarity vs Subjectivity')
axes[1].set_xlabel('Polarity')
axes[1].set_ylabel('Subjectivity')

plt.tight_layout()
plt.savefig('polarity_plots.png', dpi=150)
plt.show()
```

5.2 Sentiment Category Chart

```
import seaborn as sns

def categorize(pol):
    if pol > 0.1:  return 'Positive'
    if pol < -0.1: return 'Negative'
    return 'Neutral'

df['sentiment'] = df['polarity'].apply(categorize)

fig, axes = plt.subplots(1, 2, figsize=(13, 5))
palette   = {'Positive':'teal','Negative':'crimson','Neutral':'gray'}

# Count plot
sns.countplot(x='sentiment', data=df,
              order=['Positive','Neutral','Negative'],
              palette=palette, ax=axes[0])
axes[0].set_title('Sentiment Distribution')

# Pie chart
counts = df['sentiment'].value_counts()
axes[1].pie(counts, labels=counts.index,
            colors=[palette[l] for l in counts.index],
            autopct='%1.1f%%', startangle=90)
axes[1].set_title('Sentiment Proportions')

plt.tight_layout(); plt.savefig('sentiment.png',dpi=150); plt.show()
```

5.3 Word Cloud & Frequency

```
from collections import Counter

# Word frequency bar chart (works without wordcloud library)
all_text  = ' '.join(df['clean_text'])
word_freq = Counter(all_text.split()).most_common(20)

words  = [w for w, _ in word_freq]
counts = [c for _, c in word_freq]
```

```
plt.figure(figsize=(10, 7))
plt.barh(words, counts, color='teal', edgecolor='white')
plt.gca().invert_yaxis()
plt.title('Top 20 Most Frequent Words')
plt.xlabel('Frequency')
plt.tight_layout()
plt.savefig('word_freq.png', dpi=150)
plt.show()

# Optional: Word Cloud (requires wordcloud library)
try:
    from wordcloud import WordCloud
    wc = WordCloud(width=900, height=450, background_color='white',
                   colormap='viridis', max_words=120,
                   collocations=False)
    wc.generate_from_frequencies(dict(Counter(all_text.split())))
    plt.figure(figsize=(12, 6))
    plt.imshow(wc, interpolation='bilinear')
    plt.axis('off')
    plt.title('Word Cloud — Amazon Alexa Reviews')
    plt.savefig('wordcloud.png', dpi=150)
    plt.show()
except ImportError:
    print('wordcloud not installed — using frequency chart above')
```

6 Text Classification — Naive Bayes & SVM

Objective	Train and evaluate Naive Bayes and SVM classifiers on review data
Dataset	amazon_alexa.tsv (feedback column as label)
Libraries	scikit-learn

6.1 What is Text Classification?

Text Classification is a supervised ML task where a model learns to assign predefined labels to text documents. It trains on labeled examples and generalizes to unseen text.

Type	Description	Example
Binary	Two classes only	Spam vs Not Spam
Multi-class	Three or more classes	Sports / Tech / Politics / Health
Multi-label	Multiple labels per document	Article tagged Tech AND AI

6.2 Naive Bayes Classifier

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, classification_report

# Prepare data
X = df['clean_text']
y = df['feedback']

# Split 80% train, 20% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# Vectorize
tfidf = TfidfVectorizer(max_features=5000)
X_tr  = tfidf.fit_transform(X_train)
X_te  = tfidf.transform(X_test)

# Train Naive Bayes
nb = MultinomialNB(alpha=1.0)  # alpha = Laplace smoothing
```

```
nb.fit(X_tr, y_train)

# Predict and evaluate
y_pred = nb.predict(X_te)
print(f'Accuracy: {accuracy_score(y_test, y_pred):.4f}')
print(classification_report(y_test, y_pred))

# 5-fold cross-validation
cv_scores = cross_val_score(nb, X_tr, y_train, cv=5)
print(f'CV Mean: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}')
```

6.3 SVM Classifier

```
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline

# SVM with Pipeline (recommended approach)
svm_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=10000, ngram_range=(1,2))),
    ('svm',   LinearSVC(C=1.0, max_iter=1000))
])

svm_pipeline.fit(X_train, y_train)
y_pred_svm = svm_pipeline.predict(X_test)

print('SVM Results:')
print(f'Accuracy: {accuracy_score(y_test, y_pred_svm):.4f}')
print(classification_report(y_test, y_pred_svm))
```

6.4 Compare Both Models

```
from sklearn.linear_model import LogisticRegression

models = {
    'Naive Bayes': Pipeline([
        ('tfidf', TfidfVectorizer(max_features=5000)),
        ('nb',    MultinomialNB())
    ]),
    'SVM': Pipeline([
        ('tfidf', TfidfVectorizer(max_features=10000,ngram_range=(1,2))),
        ('svm',   LinearSVC(C=1.0))
```

```
    ]),
    'Logistic Regression': Pipeline([
        ('tfidf', TfidfVectorizer(max_features=5000)),
        ('lr',    LogisticRegression(max_iter=1000))
    ]),
}

print(f'{'Model':22} | {'Accuracy':10} | {'CV Score':10}')
print('-' * 48)
for name, model in models.items():
    model.fit(X_train, y_train)
    acc = accuracy_score(y_test, model.predict(X_test))
    cv  = cross_val_score(model, X_train, y_train, cv=5).mean()
    print(f'{name:22} | {acc:.4f}     | {cv:.4f}')
```

7 Transformers & BERT

Objective	Understand transformer architecture and use BERT for text classification
Dataset	amazon_alexa.tsv
Libraries	transformers, torch, sklearn

7.1 What are Transformers?

Transformers are deep learning architectures introduced in 2017 ("Attention Is All You Need"). They replaced RNNs for NLP tasks by processing entire sequences in parallel using a mechanism called Self-Attention.

Feature	Traditional RNN/LSTM	Transformer
Processing	Sequential (word by word)	Parallel (all words at once)
Context	Limited (forgets long context)	Full context (all positions)
Training speed	Slow (cannot parallelize)	Fast (GPU-parallelizable)
State of the art	2014–2017	2018–present
Examples	LSTM, GRU	BERT, GPT, T5, RoBERTa

7.2 How BERT Works

BERT (Bidirectional Encoder Representations from Transformers) reads text in both directions simultaneously, capturing full context for each word. It was pre-trained on Wikipedia and BookCorpus, then fine-tuned for specific tasks.

> **Key Insight**
>
> Before BERT, 'bank' in 'river bank' and 'bank account' would be represented the same way. BERT uses context from BOTH sides to understand the correct meaning of each word.

```
# Install: pip install transformers torch
from transformers import pipeline


# ── Option 1: Zero-shot classification (no training needed) ──
classifier = pipeline('text-classification',
    model='distilbert-base-uncased-finetuned-sst-2-english')
```

```
reviews = [
    'I love this Alexa, works perfectly!',
    'Terrible device, stopped working after one week.',
    'It is okay, nothing special.',
]

for review in reviews:
    result = classifier(review)[0]
    print(f'Review: {review[:50]}')
    print(f'Label : {result["label"]}  Score: {result["score"]:.3f}')
    print()
```

Fine-tuning BERT on Amazon Alexa Data

```
from transformers import (AutoTokenizer, AutoModelForSequenceClassification,
                          Trainer, TrainingArguments)
from torch.utils.data import Dataset
import torch

# ── Step 1: Tokenize the data ──────────────────────────────────
MODEL_NAME = 'distilbert-base-uncased'
tokenizer  = AutoTokenizer.from_pretrained(MODEL_NAME)

class ReviewDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.encodings = tokenizer(
            list(texts), truncation=True,
            padding=True, max_length=max_len)
        self.labels = list(labels)
    def __len__(self): return len(self.labels)
    def __getitem__(self, idx):
        item = {k: torch.tensor(v[idx])
                for k, v in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

# ── Step 2: Create datasets ────────────────────────────────────
from sklearn.model_selection import train_test_split
X_tr, X_te, y_tr, y_te = train_test_split(
    df['text'], df['feedback'],
    test_size=0.2, random_state=42)
```

```
train_dataset = ReviewDataset(X_tr, y_tr, tokenizer)
test_dataset  = ReviewDataset(X_te, y_te, tokenizer)

# ── Step 3: Load model ──────────────────────────────────────────
model = AutoModelForSequenceClassification.from_pretrained(
    MODEL_NAME, num_labels=2)

# ── Step 4: Training arguments ──────────────────────────────────
args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    evaluation_strategy='epoch',
    save_strategy='epoch',
    load_best_model_at_end=True,
    logging_dir='./logs',
)

# ── Step 5: Train ───────────────────────────────────────────────
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)
trainer.train()
print('Training complete!')
```

8 Large Language Models (LLMs)

Objective	Understand LLMs and use them via API and local inference
Dataset	amazon_alexa.tsv reviews
Libraries	transformers, openai (optional), ollama (optional)

8.1 What are LLMs?

Large Language Models are transformer-based models trained on massive datasets (hundreds of billions of tokens). They can generate text, answer questions, summarize, translate, and perform classification — all without task-specific training.

Model	Created by	Parameters	Best for
GPT-4	OpenAI	~1 trillion	General tasks, coding, reasoning
BERT	Google	110M–340M	Classification, NER, Q&A
LLaMA 3	Meta	8B–70B	Local deployment, research
Mistral	Mistral AI	7B	Efficient local inference
T5	Google	220M–11B	Text-to-text, summarization

8.2 Using a Local LLM with Hugging Face

```
from transformers import pipeline

# ── Sentiment classification using a pre-trained LLM ──────────
# Uses DistilBERT — lightweight and fast, no GPU needed
sentiment_pipe = pipeline(
    'text-classification',
    model='distilbert-base-uncased-finetuned-sst-2-english',
    return_all_scores=True
)

sample_reviews = df['verified_reviews'].dropna().head(5).tolist()
```

```
for review in sample_reviews:
    result = sentiment_pipe(review[:512])[0]  # truncate to 512 tokens
    scores = {r['label']: r['score'] for r in result}
    pred   = max(scores, key=scores.get)
    print(f'Review : {review[:60]}...')
    print(f'Positive: {scores.get("POSITIVE",0):.3f}',
          f'  Negative: {scores.get("NEGATIVE",0):.3f}')
    print(f'Prediction: {pred}')
    print()
```

Text Summarization with an LLM

```
from transformers import pipeline

# Facebook BART — good for summarization
summarizer = pipeline('summarization',
    model='facebook/bart-large-cnn',
    max_length=60, min_length=20)

# Summarize long reviews
long_reviews = df[df['word_count'] > 50]['verified_reviews'].head(3)

for review in long_reviews:
    review_text = str(review)[:1000]  # BART has token limit
    summary = summarizer(review_text)[0]['summary_text']
    print('ORIGINAL:', review_text[:150], '...')
    print('SUMMARY :', summary)
    print()
```

Zero-Shot Classification with LLM

```
from transformers import pipeline

# Zero-shot: classify without any fine-tuning
zero_shot = pipeline('zero-shot-classification',
    model='facebook/bart-large-mnli')

candidate_labels = ['positive experience','negative experience',
                    'neutral feedback','product complaint','product praise']
```

```
sample = 'I love how easy Alexa is to set up and connect to other devices'
result = zero_shot(sample, candidate_labels)


print('Text:', sample)
print('\nClassification Results:')
for label, score in zip(result['labels'], result['scores']):
    print(f'  {label:25} {score:.3f}')
```

9 Saving & Loading Trained Models

Objective	Persist trained models to disk and reload them for inference
Dataset	Model trained on amazon_alexa.tsv
Libraries	joblib, pickle, sklearn, transformers

9.1 Why Save Models?

Training can take seconds (classical ML) or hours (deep learning). Saving a trained model means you can load it instantly at any time — for a web app, API, or demo — without retraining.

Method	Best For	File size	Pros
joblib	Scikit-learn models	Small–Medium	Fast, preserves numpy arrays
pickle	Any Python object	Small–Medium	Built-in, flexible
save_model()	Keras/TF models	Large	Full model + weights
save_pretrained()	HuggingFace models	Large	Reload with tokenizer

9.2 Save & Load Scikit-learn Model

```
import joblib
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# ── Step 1: Train the model ──────────────────────────────────────
X_train, X_test, y_train, y_test = train_test_split(
    df['clean_text'], df['feedback'],
    test_size=0.2, random_state=42)


model = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=10000, ngram_range=(1,2))),
    ('svm',   LinearSVC(C=1.0))
```

```
])
model.fit(X_train, y_train)
print(f'Trained  Accuracy: {accuracy_score(y_test, model.predict(X_test)):.4f}')

# ── Step 2: Save the model ──────────────────────────────────────
joblib.dump(model, 'nlp_classifier.pkl')
print('Model saved to nlp_classifier.pkl')

# ── Step 3: Load the model ──────────────────────────────────────
loaded_model = joblib.load('nlp_classifier.pkl')
print(f'Loaded   Accuracy: {accuracy_score(y_test, loaded_model.predict(X_test)):.4f}')

# ── Step 4: Predict on new reviews ──────────────────────────────
new_reviews = [
    'Alexa is amazing and works perfectly with all my smart devices',
    'This product broke after two days. Very disappointed.',
    'It is okay, does what it promises nothing more',
]

predictions = loaded_model.predict(new_reviews)
for review, pred in zip(new_reviews, predictions):
    label = 'Positive' if pred == 1 else 'Negative'
    print(f'[{label}] {review[:60]}...')
```

9.3 Save Vectorizer Separately

```
# Save vectorizer and model separately for more control

# Save
tfidf_vec = TfidfVectorizer(max_features=10000)
X_tr_vec  = tfidf_vec.fit_transform(X_train)

from sklearn.svm import LinearSVC
svm_clf = LinearSVC(C=1.0)
svm_clf.fit(X_tr_vec, y_train)

joblib.dump(tfidf_vec, 'tfidf_vectorizer.pkl')
joblib.dump(svm_clf,   'svm_model.pkl')
print('Vectorizer and model saved separately')
```

```
# Load and use
vec_loaded = joblib.load('tfidf_vectorizer.pkl')
clf_loaded = joblib.load('svm_model.pkl')

new_text  = ['Alexa is the best assistant I have ever owned']
X_new     = vec_loaded.transform(new_text)
prediction = clf_loaded.predict(X_new)
print('Prediction:', 'Positive' if prediction[0]==1 else 'Negative')
```

9.4 Save HuggingFace Transformer Model

```
# After fine-tuning a transformer (from Section 7)

# ── Save ──────────────────────────────────────────────────────────
model.save_pretrained('./saved_bert_model')
tokenizer.save_pretrained('./saved_bert_model')
print('BERT model saved to ./saved_bert_model/')

# ── Load ──────────────────────────────────────────────────────────
from transformers import (AutoTokenizer,
                          AutoModelForSequenceClassification,
                          pipeline)

loaded_tok   = AutoTokenizer.from_pretrained('./saved_bert_model')
loaded_model = AutoModelForSequenceClassification.from_pretrained(
    './saved_bert_model')

# Create pipeline with loaded model
clf_pipe = pipeline('text-classification',
    model=loaded_model, tokenizer=loaded_tok)

result = clf_pipe('This Alexa is absolutely fantastic!')
print('Prediction:', result)
```

10 Python GUI for NLP Applications

Objective	Build a desktop GUI app for NLP review classification using tkinter
Dataset	Uses nlp_classifier.pkl saved in Section 9
Libraries	tkinter (built-in), joblib, textblob

10.1 Introduction to tkinter

tkinter is Python's built-in GUI library — no installation needed. It creates cross-platform desktop apps with buttons, text fields, labels, and more. Perfect for wrapping an NLP model in a user interface.

Widget	Purpose	Usage example
Tk()	Create main window	root = tk.Tk()
Label	Display text or image	tk.Label(root, text='Hello')
Entry	Single-line text input	tk.Entry(root, width=50)
Text	Multi-line text area	tk.Text(root, height=5)
Button	Clickable button	tk.Button(root, text='Go', command=fn)
Frame	Group widgets together	tk.Frame(root, bg='white')
StringVar	Dynamic text variable	var = tk.StringVar()

10.2 Simple Review Classifier GUI

```
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
import joblib
from textblob import TextBlob
import re


# ── Load the trained model ──────────────────────────────────────
# Make sure you ran Section 9 first to create this file
try:
    model = joblib.load('nlp_classifier.pkl')
    MODEL_READY = True
except FileNotFoundError:
    MODEL_READY = False
    print('Warning: nlp_classifier.pkl not found. Run Section 9 first.')
```

```
# ── Text cleaning function ──────────────────────────────────────
def clean_input(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    return text.strip()

# ── Prediction function ─────────────────────────────────────────
def predict_review():
    review_text = review_input.get('1.0', tk.END).strip()
    if not review_text:
        messagebox.showwarning('Empty Input',
            'Please enter a review first!')
        return

    # Sentiment via TextBlob
    blob      = TextBlob(review_text)
    polarity  = blob.sentiment.polarity
    subjectivity = blob.sentiment.subjectivity

    # ML prediction
    if MODEL_READY:
        cleaned  = clean_input(review_text)
        ml_pred  = model.predict([cleaned])[0]
        ml_label = 'Positive ✓' if ml_pred == 1 else 'Negative ✗'
    else:
        ml_label = 'Model not loaded'

    # TextBlob sentiment label
    if polarity > 0.1:  tb_label = 'Positive 😊'
    elif polarity < -0.1: tb_label = 'Negative 😞'
    else: tb_label = 'Neutral 😐'

    # Display results
    result_var.set(
        f'ML Model: {ml_label}\n'
        f'TextBlob: {tb_label}\n'
        f'Polarity: {polarity:.3f}\n'
        f'Subjectivity: {subjectivity:.3f}'
    )
```

```
    # Change result background by sentiment
    if polarity > 0.1:
        result_label.config(bg='#D4EDDA', fg='#1A5C38')
    elif polarity < -0.1:
        result_label.config(bg='#FDECEA', fg='#8B1A1A')
    else:
        result_label.config(bg='#FFF3CD', fg='#7B4F00')

def clear_all():
    review_input.delete('1.0', tk.END)
    result_var.set('Results will appear here...')
    result_label.config(bg='#F5F5F5', fg='#4A4A4A')

# ── Build the GUI ──────────────────────────────────────────────
root = tk.Tk()
root.title('NLP Review Classifier — Amazon Alexa')
root.geometry('700x580')
root.configure(bg='#1F3864')
root.resizable(True, True)

# Title bar
title_frame = tk.Frame(root, bg='#1F3864', pady=15)
title_frame.pack(fill='x')
tk.Label(title_frame, text='Amazon Alexa Review Classifier',
         font=('Calibri', 18, 'bold'), bg='#1F3864',
         fg='white').pack()
tk.Label(title_frame, text='Powered by NLP — Machine Learning + TextBlob',
         font=('Calibri', 10), bg='#1F3864',
         fg='#C8DCF0').pack()

# Main content frame
main = tk.Frame(root, bg='white', padx=20, pady=20)
main.pack(fill='both', expand=True, padx=15, pady=(0,15))

# Input section
tk.Label(main, text='Enter a product review:',
         font=('Calibri', 12, 'bold'), bg='white',
         fg='#1F3864').pack(anchor='w', pady=(0,5))

review_input = scrolledtext.ScrolledText(
    main, height=6, width=70,
    font=('Calibri', 11), wrap=tk.WORD,
```

```
    relief='solid', borderwidth=1)
review_input.pack(fill='x', pady=(0,10))

# Buttons
btn_frame = tk.Frame(main, bg='white')
btn_frame.pack(fill='x', pady=(0,15))

tk.Button(btn_frame, text='🔍 Analyze Review',
          font=('Calibri', 12, 'bold'),
          bg='#2E5FAC', fg='white',
          activebackground='#1F3864',
          relief='flat', padx=20, pady=8,
          command=predict_review).pack(side='left', padx=(0,10))

tk.Button(btn_frame, text='🗑 Clear',
          font=('Calibri', 12),
          bg='#E0E0E0', fg='#4A4A4A',
          relief='flat', padx=20, pady=8,
          command=clear_all).pack(side='left')

# Results section
tk.Label(main, text='Analysis Results:',
         font=('Calibri', 12, 'bold'), bg='white',
         fg='#1F3864').pack(anchor='w', pady=(0,5))

result_var = tk.StringVar(value='Results will appear here...')
result_label = tk.Label(
    main, textvariable=result_var,
    font=('Calibri', 12), bg='#F5F5F5', fg='#4A4A4A',
    justify='left', anchor='w',
    relief='solid', borderwidth=1,
    padx=15, pady=15, width=60)
result_label.pack(fill='x')

# Status bar
status = 'Model loaded ✓' if MODEL_READY else 'Model not loaded ✗'
tk.Label(root, text=status,
         font=('Calibri', 9),
         bg='#1F3864', fg='#C8DCF0').pack(side='bottom', pady=5)

root.mainloop()
```

10.3 Enhanced GUI with Dataset Batch Analysis

```
# Extension: Add batch analysis of the full dataset

def analyze_dataset():
    """Run ML model on entire dataset and show stats."""
    if not MODEL_READY:
        messagebox.showerror('Error', 'Load the model first!')
        return

    try:
        import pandas as pd
        df = pd.read_csv('amazon_alexa.tsv', sep='\t')
        df['clean'] = df['verified_reviews'].astype(str).apply(clean_input)
        preds = model.predict(df['clean'])

        pos = (preds == 1).sum()
        neg = (preds == 0).sum()
        total = len(preds)

        msg = (
            f'Dataset Analysis Complete!\n\n'
            f'Total reviews: {total}\n'
            f'Predicted Positive: {pos} ({pos/total*100:.1f}%)\n'
            f'Predicted Negative: {neg} ({neg/total*100:.1f}%)'
        )
        messagebox.showinfo('Dataset Results', msg)
    except Exception as e:
        messagebox.showerror('Error', str(e))

# Add this button to btn_frame:
tk.Button(btn_frame, text='📊 Analyze Dataset',
          font=('Calibri', 12),
          bg='#1A6B7C', fg='white',
          relief='flat', padx=20, pady=8,
          command=analyze_dataset).pack(side='left', padx=(10,0))
```

Project Tip

You can combine Sections 9 and 10 to create a deployable NLP application: train and save the model (Section 9), then wrap it in a GUI (Section 10). This is a complete end-to-end NLP pipeline — from raw text to a desktop application that anyone can use.

Course Summary

Knowledge Base Systems — NLP Lab Manual

Section	Topic	Key Skills
1	Introduction to NLP	Load data, explore, understand the pipeline
2	Feature Engineering	Polarity, word count, POS features
3	Data Cleaning	Regex, tokenization, stop words, lemmatization
4	Feature Extraction	BoW, TF-IDF, N-grams with sklearn
5	Data Visualization	Matplotlib, seaborn, word clouds, heatmaps
6	Text Classification	Naive Bayes, SVM, evaluation metrics
7	Transformers & BERT	HuggingFace pipeline, fine-tuning
8	LLMs	Summarization, zero-shot, sentiment via LLM
9	Saving & Loading	joblib, pickle, save_pretrained
10	Python GUI	tkinter, full desktop NLP app

Final Project

Build a complete NLP pipeline using the chosen dataset: 1. Clean the data (Section 3) 2. Extract features (Sections 2 & 4) 3. Train and evaluate a classifier (Section 6) 4. Save the model (Section 9) 5. Build a GUI app for real-time prediction (Section 10) Optional: Replace the classifier with BERT (Section 7) for state-of-the-art accuracy.