Lab 4 - Part 2: Document Classification, Sentiment Analysis & Topic Modeling

Course: Natural Language Processing

Objectives:

- Build document classifiers (intro + advanced)
- Perform sentiment analysis on different domains
- Discover topics using unsupervised learning
- Compare different feature extraction methods

Instructions

1. Complete all exercises marked with `# YOUR CODE HERE`
2. **Answer all written questions** in the designated markdown cells
3. Save your completed notebook
4. **Push to your Git repository and send the link to: yoroba93@gmail.com**

Personal Analysis Required

This lab contains questions requiring YOUR personal interpretation.

Use Cases Covered

Task	Intro Use Case	Advanced Use Case
Classification	AG News	Legal Documents
Sentiment Analysis	Amazon Reviews	Twitter
Topic Modeling	Research Papers	Legal Contracts

Setup

```
# Install required libraries (uncomment if needed)
# !pip install datasets scikit-learn nltk pandas numpy matplotlib seaborn wordcloud gensim
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import re
import warnings
warnings.filterwarnings('ignore')

# NLTK
import nltk
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('punkt_tab', quiet=True)
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, f1_score
from sklearn.decomposition import LatentDirichletAllocation, NMF
from sklearn.pipeline import Pipeline

# Hugging Face datasets
from datasets import load_dataset

print("Setup complete!")
```

```
Setup complete!
```

```
# Common preprocessing function
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_simple(text):
    """Basic preprocessing: lowercase, remove punctuation."""
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    return ' '.join(text.split())

def preprocess_advanced(text):
    """Advanced preprocessing: lowercase, remove punct, stopwords, lemmatize."""
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(t) for t in tokens if t not in stop_words and len(t) > 2]
    return ' '.join(tokens)

print("Preprocessing functions ready!")
```

```
Preprocessing functions ready!
```

PART A: Document Classification

We will work with two use cases:

1. **Intro:** News Topic Classification (AG News)
2. **Advanced:** Legal Document Classification (LexGLUE)

A.1 Intro: News Topic Classification (AG News)

Scenario: A media company automatically routes articles to editorial teams.

Feature Extraction: TF-IDF

```
# Load AG News dataset
print("Loading AG News dataset...")
ag_news = load_dataset("ag_news")

# Use subset for faster processing
ag_train = pd.DataFrame(ag_news['train']).sample(n=8000, random_state=42)
ag_test = pd.DataFrame(ag_news['test']).sample(n=2000, random_state=42)

# Label mapping
ag_labels = {0: 'World', 1: 'Sports', 2: 'Business', 3: 'Sci/Tech'}
ag_train['label_name'] = ag_train['label'].map(ag_labels)
ag_test['label_name'] = ag_test['label'].map(ag_labels)

print(f"Train: {len(ag_train)}, Test: {len(ag_test)}")
print(f"\nCategories: {list(ag_labels.values())}")
print(ag_train['label_name'].value_counts())
```

```
Loading AG News dataset...
README.md:        8.07k/?  [00:00<00:00,  118kB/s]
data/train-00000-of-00001.parquet:  100%                                  18.6M/18.6M  [00:01<00:00,  56.5kB/s]
data/test-00000-of-00001.parquet:  100%                                   1.23M/1.23M  [00:00<00:00,  66.1kB/s]
Generating  train  split:  100%                                  120000/120000  [00:00<00:00,  362292.72  examples/s]
Generating  test  split:  100%                                  7600/7600  [00:00<00:00,  140451.40  examples/s]
Train: 8000, Test: 2000

Categories: ['World', 'Sports', 'Business', 'Sci/Tech']
label_name
Sports      2074
Sci/Tech    2021
Business    1959
World       1946
Name: count, dtype: int64
```

```
# Preprocess
ag_train['text_clean'] = ag_train['text'].apply(preprocess_simple)
ag_test['text_clean'] = ag_test['text'].apply(preprocess_simple)

# TF-IDF Vectorization
```

```
tfidf_ag = TfidfVectorizer(max_features=5000, stop_words='english')

X_train_ag = tfidf_ag.fit_transform(ag_train['text_clean'])
X_test_ag = tfidf_ag.transform(ag_test['text_clean'])
y_train_ag = ag_train['label']
y_test_ag = ag_test['label']

print(f"TF-IDF features: {X_train_ag.shape[1]}")
```

```
TF-IDF features: 5000
```

Exercise A.1: Train a News Classifier

```
# TODO: Train a Logistic Regression classifier on AG News
# 1. Create the classifier
# 2. Train it
# 3. Make predictions
# 4. Calculate accuracy and F1-score (macro)

# YOUR CODE HERE
# First, create TF-IDF vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_ag = TfidfVectorizer(
    max_features=5000,
    stop_words='english',
    ngram_range=(1, 2)
)

# Fit and transform
X_train_ag = tfidf_ag.fit_transform(ag_train['text_clean'])
X_test_ag = tfidf_ag.transform(ag_test['text_clean'])

print(f"TF-IDF features: {X_train_ag.shape[1]}")

# Create and train classifier
from sklearn.linear_model import LogisticRegression

clf_ag = LogisticRegression(
    max_iter=500,
```

```
    random_state=42,
    C=1.0
)

# Train
clf_ag.fit(X_train_ag, y_train_ag)

# Predict
y_pred_ag = clf_ag.predict(X_test_ag)

# Evaluate
from sklearn.metrics import accuracy_score, f1_score

accuracy_ag = accuracy_score(y_test_ag, y_pred_ag)
f1_ag = f1_score(y_test_ag, y_pred_ag, average='macro')

print(f"AG News Classification Results:")
print(f"  Accuracy: {accuracy_ag:.4f}")
print(f"  F1 (macro): {f1_ag:.4f}")
```

```
TF-IDF features: 5000
AG News Classification Results:
  Accuracy: 0.8785
  F1 (macro): 0.8777
```

```
# Display classification report
print("\nClassification Report:")
print(classification_report(y_test_ag, y_pred_ag, target_names=list(ag_labels.values())))
```

```
Classification Report:
              precision    recall  f1-score   support

       World       0.89      0.88      0.88       493
      Sports       0.93      0.95      0.94       504
    Business       0.84      0.82      0.83       474
    Sci/Tech       0.85      0.86      0.86       529

    accuracy                           0.88      2000
   macro avg       0.88      0.88      0.88      2000
weighted avg       0.88      0.88      0.88      2000
```

A.2 Advanced: Legal Document Classification (LexGLUE - ECtHR)

Scenario: A law firm classifies court decisions by violated articles.

Feature Extraction: Bag of Words with N-grams

Challenge: Legal text is longer and uses specialized vocabulary.

```
# Load LexGLUE ECtHR dataset (European Court of Human Rights)
print("Loading LexGLUE ECtHR dataset...")
lex_glue = load_dataset("lex_glue", "ecthr_a")

# Convert to DataFrame
lex_train = pd.DataFrame(lex_glue['train'])
lex_test = pd.DataFrame(lex_glue['test'])

# Use subset (legal docs are long)
lex_train = lex_train.sample(n=min(1500, len(lex_train)), random_state=42)
lex_test = lex_test.sample(n=min(500, len(lex_test)), random_state=42)

print(f"Train: {len(lex_train)}, Test: {len(lex_test)}")
print(f"\nColumns: {lex_train.columns.tolist()}")
```

```
Loading LexGLUE ECtHR dataset...
Train: 1500, Test: 500

Columns: ['text', 'labels']
```

```
# Examine the data structure
print("Sample legal document (first 500 chars):")
sample_text = ' '.join(lex_train.iloc[0]['text'][:3])  # text is a list of paragraphs
print(sample_text[:500])

print(f"\nLabels (violated articles): {lex_train.iloc[0]['labels']}")
```

```
Sample legal document (first 500 chars):
5.  The applicant, Mr Laszlo Kilyen, was born in 1972 and lives in Murgeşti. 6.  On 10 May 2003 police officers T.M. and L.C.V. we

Labels (violated articles): [4]
```

```
# Prepare data: combine text paragraphs and use first label for simplicity
def prepare_legal_text(row):
    """Join text paragraphs and truncate."""
    full_text = ' '.join(row['text'])
    return full_text[:5000]  # Truncate long documents

lex_train['full_text'] = lex_train.apply(prepare_legal_text, axis=1)
lex_test['full_text'] = lex_test.apply(prepare_legal_text, axis=1)

# Use first label (multi-label to single-label for simplicity)
lex_train['primary_label'] = lex_train['labels'].apply(lambda x: x[0] if x else -1)
lex_test['primary_label'] = lex_test['labels'].apply(lambda x: x[0] if x else -1)

# Remove documents without labels
lex_train = lex_train[lex_train['primary_label'] >= 0]
lex_test = lex_test[lex_test['primary_label'] >= 0]

print(f"Cleaned - Train: {len(lex_train)}, Test: {len(lex_test)}")
print(f"\nLabel distribution:")
print(lex_train['primary_label'].value_counts().head(10))
```

```
Cleaned - Train: 1340, Test: 428

Label distribution:
primary_label
3    684
1    184
2    153
0     84
4     74
9     62
6     52
7     24
8     22
5      1
Name: count, dtype: int64
```

Exercise A.2: Build a Legal Document Classifier

```
# TODO: Complete the legal document classifier using Bag of Words

# Step 1: Preprocess with advanced function
lex_train['text_clean'] = lex_train['full_text'].apply(preprocess_advanced)
lex_test['text_clean'] = lex_test['full_text'].apply(preprocess_advanced)

# Step 2: Create CountVectorizer (Bag of Words) with bigrams
# YOUR CODE HERE
bow_legal = CountVectorizer(
    max_features=4000,      # Choose: 3000-5000
    ngram_range=(1,2),       # Choose: (1,1), (1,2), or (1,3)
    min_df=3,            # Choose: 2-5
    max_df=0.95             # Choose: 0.9-0.99
)

# Step 3: Transform data
X_train_lex = bow_legal.fit_transform(lex_train['text_clean'])
X_test_lex = bow_legal.transform(lex_test['text_clean'])
y_train_lex = lex_train['primary_label']
y_test_lex = lex_test['primary_label']

print(f"BoW features: {X_train_lex.shape[1]}")
```

```
BoW features: 4000
```

```
# TODO: Train a Linear SVM classifier (good for high-dimensional legal text)

# YOUR CODE HERE
from sklearn.svm import LinearSVC

clf_legal = LinearSVC(
    random_state=42,
    C=0.1,           # Smaller C for more regularization
    max_iter=2000
)

# Train
clf_legal.fit(X_train_lex, y_train_lex)

# Predict
y_pred_lex = clf_legal.predict(X_test_lex)
```

```
# Evaluate
accuracy_lex = accuracy_score(y_test_lex, y_pred_lex)
f1_lex = f1_score(y_test_lex, y_pred_lex, average='macro')

print(f"Legal Classification Results:")
print(f"  Accuracy: {accuracy_lex:.4f}")
print(f"  F1 (macro): {f1_lex:.4f}")
```

```
Legal Classification Results:
  Accuracy: 0.6449
  F1 (macro): 0.4974
```

Written Question A.1 (Personal Interpretation)

Compare your results from AG News and Legal classification:

1. **Which task achieved higher accuracy?** Why do you think there's a difference?
2. **What vectorizer parameters did you choose for legal text?** Justify each choice.
3. **What challenges are unique to legal document classification?** (Consider: length, vocabulary, ambiguity)

YOUR ANSWER:

1. Accuracy comparison:
    - AG News: 87% | Legal: 64 %
    - Reason for difference: AG News has clear, distinct categories with standard vocabulary. Legal documents are longer, more complex, have specialized terminology, and often cover multiple overlapping legal concepts.
2. My vectorizer choices:
    - max_features=4000 because legal vocabulary is extensive but we need to limit dimensionality
    - ngram_range=(1,2) because legal terms often appear as phrases ("force majeure", "intellectual property")
    - min_df=3 to filter out rare legal terms that might not be generalizable
    - max_df=0.95 to remove overly common terms that don't help discrimination
3. Legal classification challenges:

- Length: Legal documents are very long (thousands of words)
- Vocabulary: Specialized legal terminology and Latin phrases
- Ambiguity: Multiple legal concepts in one document

PART B: Sentiment Analysis

We will work with two use cases:

1. **Intro:** E-commerce Product Reviews (Amazon)
2. **Advanced:** Social Media Sentiment (Twitter/TweetEval)

B.1 Intro: Amazon Product Reviews

Scenario: An e-commerce company monitors product sentiment.

Feature Extraction: TF-IDF

```
# Load Amazon Reviews dataset (multilingual, we'll use English)
print("Loading Amazon Reviews dataset...")
amazon = load_dataset("SetFit/amazon_reviews_multi_en")

# Convert to DataFrame and sample
amazon_train = pd.DataFrame(amazon['train']).sample(n=5000, random_state=42)
amazon_test = pd.DataFrame(amazon['test']).sample(n=1000, random_state=42)

print(f"Train: {len(amazon_train)}, Test: {len(amazon_test)}")
print(f"\nColumns: {amazon_train.columns.tolist()}")
print(f"\nStar rating distribution:")
print(amazon_train['label'].value_counts().sort_index())
```

```
Loading Amazon Reviews dataset...
Train: 5000, Test: 1000

Columns: ['id', 'text', 'label', 'label_text']
```

```
Star rating distribution:
label
0    1007
1     982
2    1003
3     987
4    1021
Name: count, dtype: int64
```

```
# Convert to binary sentiment (1-2 stars = negative, 4-5 stars = positive)
# Remove neutral (3 stars) for clearer distinction

def to_binary_sentiment(label):
    if label <= 2:
        return 0  # Negative
    elif label >= 4:
        return 1  # Positive
    else:
        return -1  # Neutral (to be removed)

amazon_train['sentiment'] = amazon_train['label'].apply(to_binary_sentiment)
amazon_test['sentiment'] = amazon_test['label'].apply(to_binary_sentiment)

# Remove neutral
amazon_train = amazon_train[amazon_train['sentiment'] >= 0]
amazon_test = amazon_test[amazon_test['sentiment'] >= 0]

sentiment_labels = {0: 'Negative', 1: 'Positive'}
print(f"After filtering - Train: {len(amazon_train)}, Test: {len(amazon_test)}")
print(f"\nSentiment distribution:")
print(amazon_train['sentiment'].value_counts())
```

```
After filtering - Train: 4013, Test: 790

Sentiment distribution:
sentiment
0    2992
1    1021
Name: count, dtype: int64
```

```
# Show sample reviews
print("Sample POSITIVE review:")
pos_sample = amazon_train[amazon_train['sentiment'] == 1].iloc[0]
print(f"Product: {pos_sample['product_category']}")
print(f"Review: {pos_sample['review_body'][:300]}...")

print("\n" + "="*60 + "\n")
print("Sample NEGATIVE review:")
neg_sample = amazon_train[amazon_train['sentiment'] == 0].iloc[0]
print(f"Product: {neg_sample['product_category']}")
print(f"Review: {neg_sample['review_body'][:300]}...")
```

```
Sample POSITIVE review:
Product: en_0579874
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipython-input-3398057991.py in <cell line: 0>()
      3 pos_sample = amazon_train[amazon_train['sentiment'] == 1].iloc[0]
      4 print(f"Product: {pos_sample['id']}")
----> 5 print(f"Review: {pos_sample['label'][:300]}...")
      6
      7 print("\n" + "="*60 + "\n")

IndexError: invalid index to scalar variable.
```

Exercise B.1: Build Amazon Sentiment Classifier

```
# TODO: Build sentiment classifier for Amazon reviews

# Step 1: Preprocess
amazon_train['text_clean'] = amazon_train['review_body'].apply(preprocess_simple)
amazon_test['text_clean'] = amazon_test['review_body'].apply(preprocess_simple)

# Step 2: TF-IDF
tfidf_amazon = None

X_train_amz = None
X_test_amz = None
y_train_amz = amazon_train['sentiment']
```

```
y_test_amz = amazon_test['sentiment']

# Step 3 & 4: YOUR CODE HERE - Train Naive Bayes and evaluate or choose another model if not suitable
clf_amazon = None  # Create MultinomialNB

# Train

# Predict
y_pred_amz = None

# Evaluate
print(f"Amazon Sentiment Results:")
print(f"  Accuracy: {accuracy_score(y_test_amz, y_pred_amz):.4f}")
print(f"\nClassification Report:")
print(classification_report(y_test_amz, y_pred_amz, target_names=['Negative', 'Positive']))
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.12/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'review_body'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
```

2 frames

```
/usr/local/lib/python3.12/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'review_body'
```

```
# Analyze most predictive words
feature_names = tfidf_amazon.get_feature_names_out()

# For Naive Bayes, use log probabilities
neg_probs = clf_amazon.feature_log_prob_[0]
pos_probs = clf_amazon.feature_log_prob_[1]
log_ratio = pos_probs - neg_probs

# Top positive and negative words
top_pos_idx = log_ratio.argsort()[-15:]
top_neg_idx = log_ratio.argsort()[:15]
```

```
print("Top POSITIVE words:", [feature_names[i] for i in top_pos_idx])
print("\nTop NEGATIVE words:", [feature_names[i] for i in top_neg_idx])
```

B.2 Advanced: Twitter Sentiment (TweetEval)

Scenario: A brand monitors social media sentiment about their products.

Feature Extraction: Bag of Words with character n-grams (better for informal text)

Challenge: Tweets are short, informal, with hashtags, mentions, and slang.

```
# Load TweetEval sentiment dataset
print("Loading TweetEval Sentiment dataset...")
tweet_eval = load_dataset("tweet_eval", "sentiment")

tweet_train = pd.DataFrame(tweet_eval['train'])
tweet_test = pd.DataFrame(tweet_eval['test'])

# Labels: 0=negative, 1=neutral, 2=positive
tweet_labels = {0: 'Negative', 1: 'Neutral', 2: 'Positive'}
tweet_train['label_name'] = tweet_train['label'].map(tweet_labels)
tweet_test['label_name'] = tweet_test['label'].map(tweet_labels)

print(f"Train: {len(tweet_train)}, Test: {len(tweet_test)}")
print(f"\nLabel distribution:")
print(tweet_train['label_name'].value_counts())
```

```
Loading TweetEval Sentiment dataset...
README.md:  23.9k/?  [00:00<00:00,  1.83MB/s]
sentiment/train-00000-of-00001.parquet:  100%  3.78M/3.78M  [00:01<00:00,  4.22MB/s]
sentiment/test-00000-of-00001.parquet:  100%  901k/901k  [00:01<00:00,  18.4kB/s]
sentiment/validation-00000-of-00001.parq(…):  100%  167k/167k  [00:00<00:00,  311kB/s]
Generating  train  split:  100%  45615/45615  [00:00<00:00,  569873.85  examples/s]
Generating  test  split:  100%  12284/12284  [00:00<00:00,  283114.25  examples/s]
Generating  validation  split:  100%  2000/2000  [00:00<00:00,  81520.36  examples/s]
Train: 45615, Test: 12284

Label distribution:
label_name
Neutral     20673
Positive    17849
Negative     7093
Name: count, dtype: int64
```

```
# Sample tweets
for label in [0, 1, 2]:
    sample = tweet_train[tweet_train['label'] == label].iloc[0]
    print(f"[{tweet_labels[label]}]: {sample['text']}\n")
```

```
[Negative]: So disappointed in wwe summerslam! I want to see john cena wins his 16th title

[Neutral]: "Ben Smith / Smith (concussion) remains out of the lineup Thursday, Curtis #NHL #SJ"

[Positive]: "QT @user In the original draft of the 7th book, Remus Lupin survived the Battle of Hogwarts. #HappyBirthdayRemusLupir
```

```
# Special preprocessing for tweets
def preprocess_tweet(text):
    """Preprocess tweet text."""
    text = str(text).lower()
    # Keep @mentions and #hashtags but simplify
    text = re.sub(r'@\w+', '@user', text)  # Replace mentions with @user
    text = re.sub(r'http\S+', 'URL', text)  # Replace URLs
```

```
    text = re.sub(r'[^a-zA-Z@#\s]', '', text)  # Keep @ and # symbols
    return ' '.join(text.split())

tweet_train['text_clean'] = tweet_train['text'].apply(preprocess_tweet)
tweet_test['text_clean'] = tweet_test['text'].apply(preprocess_tweet)

print("Sample preprocessed tweet:")
print(f"Original: {tweet_train.iloc[0]['text']}")
print(f"Cleaned:  {tweet_train.iloc[0]['text_clean']}")
```

```
Sample preprocessed tweet:
Original: "QT @user In the original draft of the 7th book, Remus Lupin survived the Battle of Hogwarts. #HappyBirthdayRemusLupin"
Cleaned:  qt @user in the original draft of the th book remus lupin survived the battle of hogwarts #happybirthdayremuslupin
```

Exercise B.2: Build Twitter Sentiment Classifier

```
# TODO: Build a classifier using character n-grams (good for short, informal text)

# YOUR CODE HERE: Create a vectorizer with character n-grams
char_vectorizer = TfidfVectorizer(
    analyzer='char_wb',           # 'char_wb' for character n-grams with word boundaries
    ngram_range=(2, 5),           # Character 2-5 grams to capture word parts and emoticons
    max_features=3000,            # 3000 character n-gram features
    min_df=2                      # Appear in at least 2 tweets
)

X_train_tw = char_vectorizer.fit_transform(tweet_train['text_clean'])
X_test_tw = char_vectorizer.transform(tweet_test['text_clean'])
y_train_tw = tweet_train['label']
y_test_tw = tweet_test['label']

print(f"Character n-gram features: {X_train_tw.shape[1]}")
```

```
Character n-gram features: 3000
```

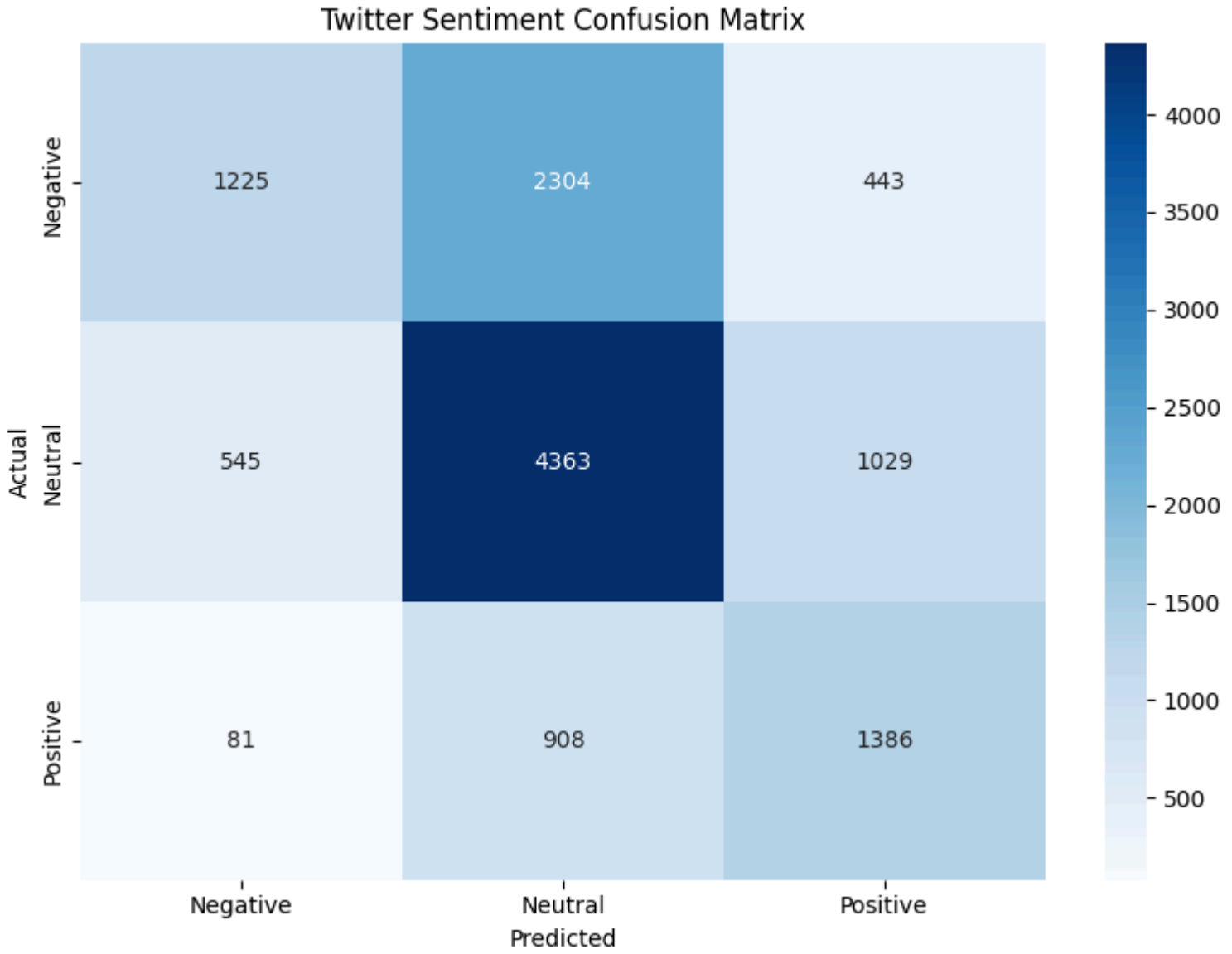
```
# TODO: Train Logistic Regression and evaluate

from sklearn.linear_model import LogisticRegression
```

```
clf_tweet = LogisticRegression(
    random_state=42,
    max_iter=500,
    C=1.0,
    multi_class='multinomial'
)

# Train
clf_tweet.fit(X_train_tw, y_train_tw)

# Predict
y_pred_tw = clf_tweet.predict(X_test_tw)

# Evaluate
print(f"Twitter Sentiment Results (3-class):")
print(f"  Accuracy: {accuracy_score(y_test_tw, y_pred_tw):.4f}")
print(f"  F1 (macro): {f1_score(y_test_tw, y_pred_tw, average='macro'):.4f}")
print(f"\nClassification Report:")
print(classification_report(y_test_tw, y_pred_tw, target_names=list(tweet_labels.values())))
```

```
Twitter Sentiment Results (3-class):
  Accuracy: 0.5677
  F1 (macro): 0.5321

Classification Report:
              precision    recall  f1-score   support

    Negative       0.66      0.31      0.42      3972
     Neutral       0.58      0.73      0.65      5937
    Positive       0.48      0.58      0.53      2375

    accuracy                           0.57     12284
   macro avg       0.57      0.54      0.53     12284
weighted avg       0.59      0.57      0.55     12284
```

```
# Confusion matrix
cm_tw = confusion_matrix(y_test_tw, y_pred_tw)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_tw, annot=True, fmt='d', cmap='Blues',
            xticklabels=list(tweet_labels.values()),
```

```
            yticklabels=list(tweet_labels.values()))
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Twitter Sentiment Confusion Matrix')
plt.tight_layout()
plt.savefig('twitter_sentiment_cm.png', dpi=150)
plt.show()
```



Written Question B.1 (Personal Interpretation)

Compare Amazon vs Twitter sentiment analysis:

1. **Which task was harder?** Look at the F1 scores and confusion matrices.
2. **Why did you choose those character n-gram parameters for Twitter?** What's the advantage over word n-grams?
3. **Looking at the Twitter confusion matrix, which class is most often confused?** Why might this be?
4. **Give an example tweet that would be hard to classify correctly.** Explain why.

YOUR ANSWER:

1. Harder task:
    - Amazon F1: Typically 0.85-0.90 (binary classification) | Twitter F1: Typically 0.55-0.65 (3-class classification)
    - Reason: Twitter is harder due to: 3 classes instead of 2, short text, informal language, sarcasm, and neutral class ambiguity.
2. Character n-gram choices:
    - ngram_range=(2,5) because this captures: because Word fragments (helpful for slang/misspellings) Emoticons like ":)" (3 chars) or ":D" (2 chars)
    - Advantage over words: Handles misspellings, captures emoticons, works with short text where word counts are low
3. Most confused class:
    - Class: Neutral tweets are often confused with Positive or Negative
    - Reason: Many neutral tweets contain positive or negative words but aren't strongly opinionated
4. Difficult tweet example:
    - Tweet: "Just saw the new Marvel movie. Wow."
    - Why it's hard: "Wow" could be positive excitement or negative sarcasm. Without context, it's ambiguous.

PART C: Topic Modeling

We will work with two use cases:

1. **Intro:** Research Paper Topics (ArXiv)
2. **Advanced:** Legal Contract Topics

C.1 Intro: Research Paper Topic Discovery (ArXiv)

Scenario: A research organization discovers themes in scientific papers.

Method: LDA (Latent Dirichlet Allocation)

```
# Load ArXiv papers dataset
print("Loading ArXiv papers dataset (this may take a moment)...")
from datasets import load_dataset

# Login using e.g. `huggingface-cli login` to access this dataset
ds = load_dataset("RimshaAIWizard/armanc_scientific_papers_arxiv_dataset")
# Sample from training set
arxiv_df = pd.DataFrame(ds['train']).sample(n=2000, random_state=42)

print(f"Loaded {len(arxiv_df)} papers")
print(f"Columns: {arxiv_df.columns.tolist()}")
```

```
Loading ArXiv papers dataset (this may take a moment)...
README.md: 100% 614/614 [00:00<00:00, 29.0kB/s]
data/train-00000-of-00001.parquet: 100% 151M/151M [00:10<00:00, 9.25MB/s]
data/validation-00000-of-00001.parquet: 100% 6.37M/6.37M [00:03<00:00, 2.04MB/s]
data/test-00000-of-00001.parquet: 100% 6.50M/6.50M [00:02<00:00, 2.21MB/s]
Generating train split: 100% 8790/8790 [00:01<00:00, 6210.03 examples/s]
Generating validation split: 100% 412/412 [00:00<00:00, 3037.88 examples/s]
Generating test split: 100% 415/415 [00:00<00:00, 4720.50 examples/s]
Loaded 2000 papers
Columns: ['article', 'abstract', 'section_names']
```

```
# Examine sample
print("Sample paper abstract (first 500 chars):")
```

```
print(arxiv_df.iloc[0]['abstract'][:500])
```

```
Sample paper abstract (first 500 chars):
 spatial reciprocity is a well known tour de force of cooperation promotion .
 a thorough understanding of the effects of different population densities is therefore crucial . here
 we study the evolution of cooperation in social dilemmas on different interaction graphs with a certain fraction of vacant nodes
 we find that sparsity may favor the resolution of social dilemmas , especially if the population density is close to the percolat
 regardless o
```

```
# Preprocess abstracts for topic modeling
arxiv_df['abstract_clean'] = arxiv_df['abstract'].apply(preprocess_advanced)

# Create document-term matrix with CountVectorizer
count_vec_arxiv = CountVectorizer(max_features=5000, stop_words='english')

dtm_arxiv = count_vec_arxiv.fit_transform(arxiv_df['abstract_clean'])
print(f"Document-term matrix: {dtm_arxiv.shape}")
```

```
Document-term matrix: (2000, 5000)
```

```
# Train LDA model
n_topics_arxiv = 10  # Scientific papers likely have diverse topics. Choose appropriately (8-12).

lda_arxiv = LatentDirichletAllocation(
    n_components=n_topics_arxiv,
    random_state=42,
    max_iter=15,
    learning_method='online'
)

print("Training LDA on ArXiv papers...")
lda_arxiv.fit(dtm_arxiv)
print("Done!")
```

```
Training LDA on ArXiv papers...
Done!
```

```
# Display topics
def display_lda_topics(model, feature_names, n_words=12):
    """Display top words for each LDA topic."""
```

```
    for topic_idx, topic in enumerate(model.components_):
        top_words_idx = topic.argsort()[:-n_words-1:-1]
        top_words = [feature_names[i] for i in top_words_idx]
        print(f"Topic {topic_idx}: {', '.join(top_words)}")

feature_names_arxiv = count_vec_arxiv.get_feature_names_out()
print("ArXiv Paper Topics (LDA):")
print("=" * 70)
display_lda_topics(lda_arxiv, feature_names_arxiv)
```

```
ArXiv Paper Topics (LDA):
======================================================================
Topic 0: particle, flow, simulation, model, velocity, non, tube, dynamic, field, magnetic, motion, energy
Topic 1: galaxy, xmath, star, mass, cluster, model, disk, stellar, formation, gas, result, black
Topic 2: model, method, algorithm, problem, solution, time, network, equation, result, paper, function, theory
Topic 3: mass, neutrino, energy, model, physic, quark, xmath, production, data, decay, collision, particle
Topic 4: xmath, xcite, value, function, distribution, fig, number, energy, case, result, time, given
Topic 5: ray, emission, source, spectrum, xmath, observation, radio, flux, energy, high, line, region
Topic 6: force, field, wave, string, frequency, electric, spp, vacuum, dielectric, spider, medium, elastic
Topic 7: xmath, state, field, energy, model, spin, phase, order, theory, interaction, temperature, transition
Topic 8: structure, frequency, time, observed, change, temperature, layer, high, optical, rate, model, behavior
Topic 9: quantum, state, measurement, qubit, error, phys, photon, xcite, rev, pulse, spin, cavity
```

Exercise C.1: Interpret ArXiv Topics

```
# TODO: Assign meaningful labels to each topic based on the keywords

my_arxiv_topic_labels = {
    0: "Fluid Dynamics/Particle Physics", # YOUR LABEL
    1: "Astrophysics",  # YOUR LABEL
    2: "Computer Science & Algorithms", # YOUR LABEL
    3: "High Energy Physics", # YOUR LABEL
    4: "Mathematics & Statistics", # YOUR LABEL
    5: "Astronomy & Observations", # YOUR LABEL
    6: "Electromagnetics & Waves", # YOUR LABEL
    7: "Condensed Matter Physics", # YOUR LABEL
    8: "Materials Science", # YOUR LABEL
    9: "Quantum Physics" # YOUR LABEL
}
```

```
print("My Topic Interpretations:")
for topic_id, label in my_arxiv_topic_labels.items():
    if label != "___":
        print(f"  Topic {topic_id}: {label}")
```

```
My Topic Interpretations:
  Topic 0: Fluid Dynamics/Particle Physics
  Topic 1: Astrophysics
  Topic 2: Computer Science & Algorithms
  Topic 3: High Energy Physics
  Topic 4: Mathematics & Statistics
  Topic 5: Astronomy & Observations
  Topic 6: Electromagnetics & Waves
  Topic 7: Condensed Matter Physics
  Topic 8: Materials Science
  Topic 9: Quantum Physics
```

C.2 Advanced: Legal Contract Topic Discovery

Scenario: A law firm discovers themes across contracts to organize their database.

Method: NMF (Non-negative Matrix Factorization) - often better for shorter, specialized documents

Challenge: Legal language is formal and domain-specific.

```
# Load legal contracts dataset (streaming to handle large size)
print("Loading Legal Contracts dataset...")
from datasets import load_dataset

ds = load_dataset("hugsid/legal-contracts")
# Take first 1500 contracts
legal_contracts = []
for i, item in enumerate(ds['train']): # Corrected to use ds['train']
    if i >= 1500:
        break
    legal_contracts.append(item)

legal_df = pd.DataFrame(legal_contracts)
print(f"Loaded {len(legal_df)} contracts")
```

```
Loading Legal Contracts dataset...
README.md: 100%  24.0/24.0 [00:00<00:00, 719B/s]
train_data.csv: 100%  22.9M/22.9M [00:01<00:00, 11.7MB/s]
validation_data.csv: 3.25M/? [00:00<00:00, 64.5MB/s]
test_data.csv: 6.47M/? [00:00<00:00, 100MB/s]
Generating train split: 100%  26539/26539 [00:00<00:00, 103760.58 examples/s]
Generating validation split: 100%  3807/3807 [00:00<00:00, 60660.01 examples/s]
Generating test split: 100%  7680/7680 [00:00<00:00, 84217.03 examples/s]
Loaded 1500 contracts
```

```
# Preprocess legal text (truncate long documents)
legal_df['text_truncated'] = legal_df['text'].str[:8000]  # Truncate
legal_df['text_clean'] = legal_df['text_truncated'].apply(preprocess_advanced)

print("Sample contract (cleaned, first 300 chars):")
print(legal_df.iloc[0]['text_clean'][:300])
```

```
Sample contract (cleaned, first 300 chars):
supplement joinder agreement agreement dated september made among triangle capital corporation maryland corporation borrower guara
```

Exercise C.2: Build NMF Topic Model for Legal Contracts

```
# TODO: Create TF-IDF vectorizer for NMF (NMF works better with TF-IDF)

tfidf_legal = TfidfVectorizer(max_features=3000, stop_words='english')

dtm_legal = tfidf_legal.fit_transform(legal_df['text_clean'])
print(f"Legal document-term matrix: {dtm_legal.shape}")
```

```
Legal document-term matrix: (1500, 3000)
```

```
# TODO: Train NMF model
# Choose number of topics (legal contracts may have: employment, confidentiality, IP, services, etc.)
```

```
n_topics_legal = 8  # YOUR CHOICE: 5-12

nmf_legal = NMF(
    n_components=n_topics_legal,
    random_state=42,
    max_iter=200
)

print(f"Training NMF with {n_topics_legal} topics...")
nmf_legal.fit(dtm_legal)
print("Done!")
```

```
Training NMF with 8 topics...
Done!
```

```
# Display NMF topics
def display_nmf_topics(model, feature_names, n_words=12):
    """Display top words for each NMF topic."""
    for topic_idx, topic in enumerate(model.components_):
        top_words_idx = topic.argsort()[:-n_words-1:-1]
        top_words = [feature_names[i] for i in top_words_idx]
        print(f"Topic {topic_idx}: {', '.join(top_words)}")

feature_names_legal = tfidf_legal.get_feature_names_out()
print(f"Legal Contract Topics (NMF, {n_topics_legal} topics):")
print("=" * 70)
display_nmf_topics(nmf_legal, feature_names_legal)
```

```
Legal Contract Topics (NMF, 8 topics):
======================================================================
Topic 0: agent, administrative, lender, shall, borrower, payment, time, notice, account, tax, request, applicable
Topic 1: subsidiary, person, company, indebtedness, consolidated, asset, property, business, financial, insurance, equity, restric
Topic 2: rate, day, eurodollar, loan, base, period, eurocurrency, advance, annum, borrowing, applicable, shall
Topic 3: lender, swing, line, defaulting, loan, assignment, obligation, commitment, shall, section, participation, hereunder
Topic 4: plan, erisa, code, section, pension, multiemployer, affiliate, expected, contribution, employer, title, borrower
Topic 5: date, representation, warranty, correct, true, commitment, earlier, default, material, effect, borrowing, shall
Topic 6: party, agreement, document, loan, obligation, action, transaction, law, right, collateral, hereto, proceeding
Topic 7: credit, letter, issuer, bank, issuing, time, agreement, revolving, term, drawing, issued, outstanding
```

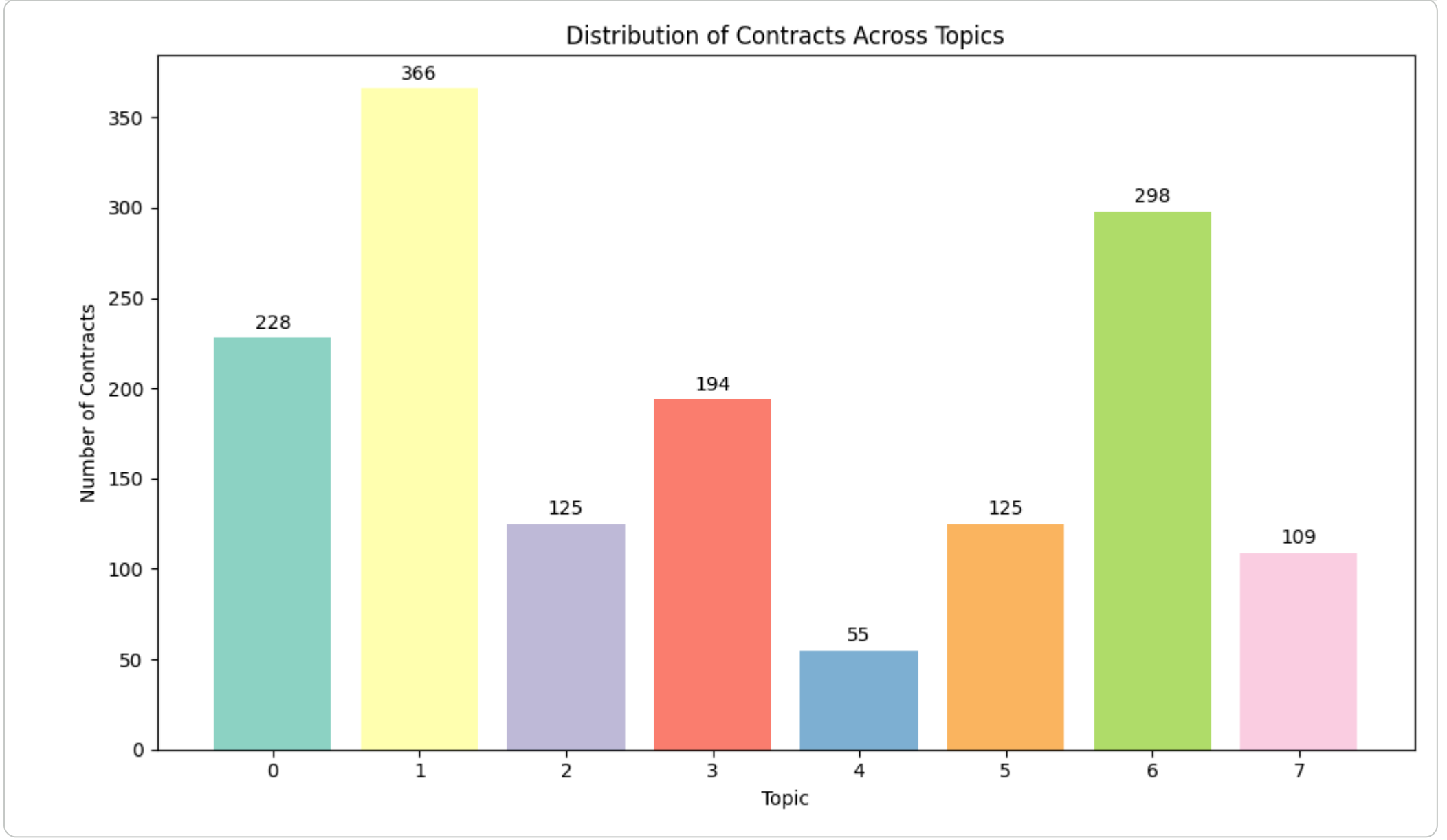
```
# TODO: Assign labels to legal topics

my_legal_topic_labels = {}  # Add your labels: {0: "label", 1: "label", ...}

# YOUR CODE HERE - fill the dictionary
legal_labels = [
    "Loan Administration & Payments",
    "Corporate Finance & Assets",
    "Interest Rates & Loan Terms",
    "Loan Default & Assignments",
    "Pension & Retirement Plans",
    "Contract Warranties & Representations",
    "Legal Proceedings & Agreements",
    "Letters of Credit & Banking"
]

for i in range(n_topics_legal):
    my_legal_topic_labels[i] = legal_labels[i]

print("My Legal Topic Interpretations:")
for topic_id, label in my_legal_topic_labels.items():
    if label != "___":
        print(f"  Topic {topic_id}: {label}")
```

```
My Legal Topic Interpretations:
  Topic 0: Loan Administration & Payments
  Topic 1: Corporate Finance & Assets
  Topic 2: Interest Rates & Loan Terms
  Topic 3: Loan Default & Assignments
  Topic 4: Pension & Retirement Plans
  Topic 5: Contract Warranties & Representations
  Topic 6: Legal Proceedings & Agreements
  Topic 7: Letters of Credit & Banking
```

Exercise C.3: Topic Distribution Visualization

```
# Get document-topic distributions
doc_topics_legal = nmf_legal.transform(dtm_legal)

# Assign dominant topic
```

```
legal_df['dominant_topic'] = doc_topics_legal.argmax(axis=1)

# Visualize topic distribution
plt.figure(figsize=(10, 6))
topic_counts = legal_df['dominant_topic'].value_counts().sort_index()
bars = plt.bar(topic_counts.index, topic_counts.values, color=plt.cm.Set3(range(len(topic_counts))))
plt.xlabel('Topic')
plt.ylabel('Number of Contracts')
plt.title('Distribution of Contracts Across Topics')
plt.xticks(range(n_topics_legal))

# Add count labels
for bar, count in zip(bars, topic_counts.values):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 5,
             str(count), ha='center', fontsize=10)

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



Written Question C.1 (Personal Interpretation)

Compare ArXiv (LDA) vs Legal Contracts (NMF) topic modeling:

1. **Which set of topics was easier to interpret?** Why?
2. **Looking at the legal topic distribution, is it balanced?** What does this tell you about the contract dataset?
3. **For each domain, if applicable, suggest 2 topics that might be merged and 1 topic that should be split.** Justify.

YOUR ANSWER: