中北大学软件学院

实 验 报 告

专　　业：	软件工程
方　　向：	人工智能
课程名称：	机器学习实践
班　　级：	22130418
学　　号：	2213041835、2213041816
姓　　名：	吕炳荣、刘晓峰
辅导教师：	程晓鼎

2024 年 3 月 制

成绩：____________

实验时间	2024 年 4 日 13 时 14 至 18 时	学时数	4 学时

1. 实验名称

法律裁判文书中的案情要素贝叶斯分类

2. 实验目的

理解贝叶斯分类原理，探索朴素贝叶斯分类器在文本分类中的应用。

3.实验内容

1、文本分类的数据预处理：文本分词、文本量化以及量化文本的重新组织。

2、以“中国裁判文书网”公开的有关婚姻家庭领域的 2665 条裁判文书为例，基于文书句子文本和每个句子对应的要素标签（多分类），探索朴素贝叶斯分类器在文本分类中的应用。

4. 实验原理或流程图

一个完整的中文文本分类系统通常由如下几个功能模块：

(1)文本预处理：文本预处理是对文档进行分词，去除停用词，其中中文分词是文本预处理的首要步骤。

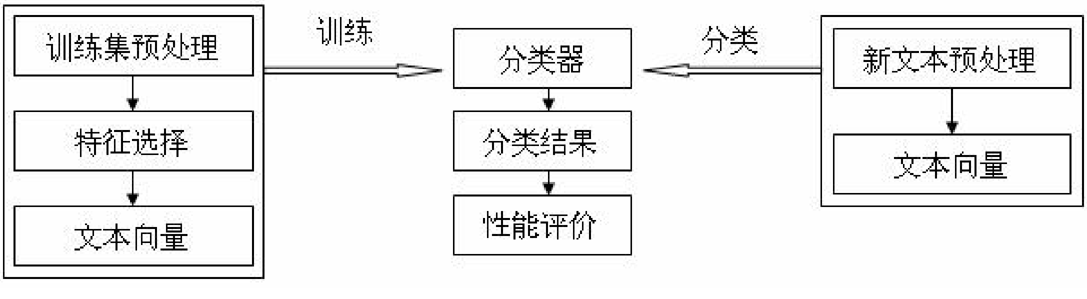
(2) 文本表示：文本表示是文本分类的基础。要将计算机技术应用到文本分类上，必须把文档转化为计算机容易处理的表示形式。目前使用最普遍的文本表示方式是向量空间模型。

(3) 文本特征选择：特征选择的目的是为了维数约简，从文档中抽取出若干最有利于文本分类的特征项。

(4)特征权重计算：特征权重是用于衡量某个特征项在文档表示中的重要程度或者区分能力的强弱。

(5)分类器学习训练：分类器学习训练的目的是建立分类器，是文本分类的核心问题。利用一定的学习算法对训练样本集进行统计学习，估算出分类器的各个参数，从而建立出对训练集进行学习训练的自动分类器。

(6)测试与评价：利用学习训练阶段建立的分类器，对测试集文档进行分类测试。在完成训练和测试后，选择合适的评价指标对分类器的性能进行评价。如果分类性能不符合要求，需要返回前面步骤，重新再做。



5. 实验过程或源代码

```
import json
import jieba
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.naive_bayes import MultinomialNB
# --- Multi-label specific imports ---
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import hamming_loss, accuracy_score as multilabel_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import classification_report
# --- Other necessary imports ---
from sklearn.pipeline import Pipeline
from wordcloud import WordCloud
import re
import warnings
import os
import matplotlib.font_manager as fm # Font manager
import traceback # For detailed error printing

# --- Configuration & Settings ---
warnings.filterwarnings('ignore') # Suppress common warnings

# --- Font Setup for Matplotlib and WordCloud ---
FONT_PATH_WC = None # Global variable for wordcloud font path
try:
    plt.rcParams['font.sans-serif'] = ['SimHei'] # Prioritize SimHei for plots
    plt.rcParams['axes.unicode_minus'] = False
    FONT_PATH_WC = fm.findfont(fm.FontProperties(family='SimHei'))
    print(f"成功设置 Matplotlib 字体为 SimHei。
```

```
WordCloud 字体路径: {FONT_PATH_WC}")
except Exception:
    print("警告: SimHei 字体未找到或设置失败。尝试查找其他中文字体...")
    # Fallback font search logic
    font_names = ['Microsoft YaHei', 'Heiti SC', 'PingFang SC', 'Noto Sans CJK SC', 'WenQuanYi Micro Hei', 'SimSun']
    found_font = False
    for font_name in font_names:
        try:
            font_prop = fm.FontProperties(family=font_name)
            font_path = fm.findfont(font_prop)
            if font_path:
                plt.rcParams['font.sans-serif'] = [font_name]
                plt.rcParams['axes.unicode_minus'] = False
                FONT_PATH_WC = font_path
                print(f"找到并设置可用字体: {font_name} ({FONT_PATH_WC})")
                found_font = True
                break
        except:
            continue
    if not found_font:
        print("警告: 未找到推荐的中文字体。绘图和词云可能无法正确显示中文。")

# --- File Paths ---
# !! IMPORTANT: Make sure these filenames match your actual files !!
DATA_FILE = '离婚诉讼文本.json'
STOPWORDS_FILE = '停用词表.txt' # Set to None if you don't have/want to use one
```

```
# --- Helper Function: Load Stopwords ---
def load_stopwords(filepath):
    """Loads stopwords from a text file."""
    stopwords = set()
    if filepath is None or not os.path.exists(filepath):
        print(f"警告: 停用词文件路径 '{filepath}' 无效或
文件不存在。将在没有自定义停用词的情况下继续。")
        return stopwords
    try:
        with open(filepath, 'r', encoding='utf-8') as f:
            stopwords = {line.strip() for line in f if
line.strip()}
        print(f"从 {filepath} 加载了 {len(stopwords)} 个
停用词。")
    except Exception as e:
        print(f"加载停用词时出错: {e}")
    return stopwords

# --- Helper Function: Robust Data Loading ---
def load_data(filepath):
    """
    Loads data from a JSON file. Handles cases where:
    1. The entire file is a single valid JSON array.
    2. Each line contains a valid JSON array of records.
    3. Each line contains a single valid JSON record.
    """
    all_records = []
    line_num = 0
    successful_records = 0
    errors_parsing_line = 0
    invalid_records_in_list = 0

    abs_path = os.path.abspath(filepath)
    print(f"Attempting to load data from: {abs_path}")

    if not os.path.exists(filepath):
        print(f"错误: 文件未找到 at path: {abs_path}")
        return None
```

```
try:
    with open(filepath, 'r', encoding='utf-8') as f:
        # --- Strategy 1: Try loading the whole file as one JSON list ---
        try:
            print("尝试将整个文件作为单个 JSON 列表加载...")
            f.seek(0) # Ensure reading from start
            entire_data = json.load(f)
            if isinstance(entire_data, list):
                print(f"成功将整个文件作为列表加载。包含 {len(entire_data)} 个潜在记录。")
                for i, record in enumerate(entire_data):
                    # Validate record structure and label type
                    if isinstance(record, dict) and 'labels' in record and 'sentence' in record and isinstance(record.get('labels'), list):
                        all_records.append(record)
                        successful_records += 1
                    else:
                        print(f"警告: 完整加载的数据中，索引 {i} 处记录格式无效(非字典/缺键/标签非列表)，已跳过。记录片段:{str(record)[:100]}...")
                        invalid_records_in_list += 1
                print(f"从完整加载的数据中验证并添加了 {successful_records} 条记录。")
                # If loaded successfully this way, no need for line-by-line
                if successful_records > 0:
```

```
                                        print("已通过完整文件加载方式成功获取数据。")
                                        # Proceed to DataFrame creation below
                            else:
                                        print("警告: 整个文件加载成功，但根元素不是列表。将尝试逐行解析。")
                                        f.seek(0) # Reset for line-by-line attempt
                                        # Fall through to line-by-line parsing
                   except json.JSONDecodeError as e_full:
                            print(f"将整个文件作为 JSON 列表加载失败: {e_full}")
                            print("将尝试逐行解析 JSON...")
                            f.seek(0) # IMPORTANT: Reset file pointer

                            # --- Strategy 2: Parse line by line ---
                            for line in f:
                                    line_num += 1
                                    line = line.strip()
                                    if not line: continue

                                    try:
                                            line_data = json.loads(line)

                                            # CASE A: Line contains a LIST of records
                                            if isinstance(line_data, list):
                                                    for record in line_data:
                                                            if isinstance(record, dict) and 'labels' in record and 'sentence' in record and isinstance(record.get('labels'), list):
```

```
all_records.append(record)
successful_records += 1
else:
print(f"警告：第 {line_num} 行列表中的记录格式无效，已跳过。记录片段：{str(record)[:100]}...")
invalid_records_in_list += 1
# CASE B: Line contains a SINGLE record (dictionary)
elif isinstance(line_data, dict):
if 'labels' in line_data and 'sentence' in line_data and isinstance(line_data.get('labels'), list):
all_records.append(line_data)
successful_records += 1
else:
print(f"警告：第 {line_num} 行的单个记录格式无效，已跳过。记录片段：{str(line_data)[:100]}...")
invalid_records_in_list += 1
else:
print(f"警告：第 {line_num} 行解析为未知类型 ({type(line_data)})，已跳过。内容：{line[:100]}...")
errors_parsing_line += 1
except json.JSONDecodeError:
```

```
                    print(f"警告: 第 {line_num} 行无法解析为 JSON，已跳过。内容: {line[:100]}...")
                    errors_parsing_line += 1

    # --- Summary after reading ---
    print("\n文件读取和解析完成。")
    print(f"总共成功加载并验证了 {successful_records} 条记录。")
    if errors_parsing_line > 0:
        print(f"有 {errors_parsing_line} 行无法被解析为 JSON。")
    if invalid_records_in_list > 0:
        print(f"有 {invalid_records_in_list} 个列表内或单个记录因格式无效被跳过。")

    if not all_records:
        print("错误: 未能从文件中收集到任何有效数据记录。请仔细检查文件格式。")
        return None

    # --- Create DataFrame ---
    df = pd.DataFrame(all_records)

    print(f"\n成功创建 DataFrame。")
    print(f"数据集形状: {df.shape}")
    if 'labels' not in df.columns or 'sentence' not in df.columns:
        print("错误: 创建的 DataFrame 中缺少 'labels' 或 'sentence' 列。")
```

```
                return None

            print("\n 数据样本 (前 5 行):")
            print(df.head())

            # Statistics
            all_labels_flat = [label for sublist in df['labels'] if
isinstance(sublist, list) for label in sublist]
            if not all_labels_flat:
                print("警告：数据中未找到任何标签。")
            else:
                label_counts                                      =
pd.Series(all_labels_flat).value_counts()
                print("\n 独立标签出现次数统计:")
                print(label_counts)

                df['label_count'] = df['labels'].apply(lambda x:
len(x) if isinstance(x, list) else 0)
                print("\n 句子标签数量分布:")

print(df['label_count'].value_counts().sort_index())
                df = df.drop(columns=['label_count']) # Drop
temporary column

            return df

        except FileNotFoundError:
            print(f"错误: 文件未找到 at path: {abs_path}") #
Should have been caught earlier
            return None
        except Exception as e:
            print(f"加载数据时发生未预料的错误: {e}")
            traceback.print_exc()
            return None

# --- Helper Function: Preprocess Text ---
def preprocess_text(text, stopwords_set):
    """Preprocesses a single text: cleans, segments, removes
```

```
stopwords."""
    if not isinstance(text, str): return ""
    # Keep only Chinese characters
    text = re.sub(r'[^\u4e00-\u9fa5]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    if not text: return ""
    # Segment using jieba
    words = jieba.lcut(text)
    # Filter out stopwords and single-character words
    filtered_words = [word for word in words if word not in
stopwords_set and len(word) > 1]
    return ' '.join(filtered_words)

# --- Helper Function: Plot Word Clouds ---
def plot_word_clouds_multilabel(df_wc, label_col, text_col,
label_names, num_labels_to_plot=8):
    """Plots word clouds for the most frequent labels in
multi-label data."""
    if label_names is None or label_col not in df_wc.columns
or text_col not in df_wc.columns:
        print("错误：绘制词云需要 label_names,
label_col, text_col。")
        return
    if df_wc.empty:
        print("错误：用于绘制词云的 DataFrame 为空。")
        return

    try:
        # Calculate label frequencies from the binarized
list/array column
        binarized_matrix =
np.array(df_wc[label_col].tolist())
        label_frequencies =
pd.Series(binarized_matrix.sum(axis=0),
index=label_names).sort_values(ascending=False)
    except Exception as e:
        print(f"错误：计算标签频率时出错。确保
'{label_col}' 包含二值化列表/数组。错误: {e}")
        return
```

```
    top_labels_indices                                      =
label_frequencies.head(num_labels_to_plot).index
    num_plots = len(top_labels_indices)
    if num_plots == 0:
        print("没有找到足够的标签来绘制词云图。")
        return

    cols = 2
    rows = (num_plots + cols - 1) // cols
    plt.figure(figsize=(16, 6 * rows))
    print(f"\n 正在为频率最高的 {num_plots} 个标签生成
词云图...")
    plot_count = 0

    for label_name in top_labels_indices:
        try:
            label_index                                     =
list(label_names).index(label_name) # Find index of the label
            # Get indices of rows where this label is
present
            subset_indices = np.where(binarized_matrix[:,
label_index] == 1)[0]
            #    Use    DataFrame's    original    index
corresponding to these numpy indices
            original_df_indices                             =
df_wc.index[subset_indices]
            # Get text using the original DataFrame indices
            subset_text                  =                  '
'.join(df_wc.loc[original_df_indices, text_col].astype(str))

            if not subset_text.strip():
                print(f"跳过标签 '{label_name}' 的词
云图生成，因为没有有效的文本内容。")
                continue

            wordcloud                                       =
WordCloud(font_path=FONT_PATH_WC, # Use detected font
path
```

```
background_color='white', width=800, height=400,

collocations=False, max_words=100).generate(subset_text)
                plot_count += 1
                plt.subplot(rows, cols, plot_count)
                plt.imshow(wordcloud, 
interpolation='bilinear')
                plt.axis('off')
                plt.title(f"包含标签: {label_name}

({int(label_frequencies[label_name])} 次)", fontsize=12)


            except Exception as e:
                print(f"为标签 '{label_name}' 生成词云时发

生错误: {e}")
                # Optionally, try without specifying font path 
as a fallback
                try:
                    print(f"尝试不指定字体为标签

'{label_name}' 生成词云...")
                    wordcloud = 
WordCloud(background_color='white', width=800, height=400,

collocations=False, max_words=100).generate(subset_text)
                    plot_count += 1 # Increment even if 
fallback plot shown
                    plt.subplot(rows, cols, plot_count) # 
Reuse plot slot
                    plt.imshow(wordcloud, 
interpolation='bilinear')
                    plt.axis('off')
                    plt.title(f"包含标签: {label_name}

({int(label_frequencies[label_name])} 次) [默认字体]", 
fontsize=10)
                except Exception as e2:
```

```
                    print(f"    使用默认字体生成词云仍然
失败: {e2}")


    if plot_count == 0: print("警告：未能成功生成任何词云
图。")
    else:
        plt.tight_layout(pad=3.0)
        plt.show()

# --- Helper Function: Run Multi-Label Experiment ---
def run_multilabel_experiment(X_train, y_train_bin, X_test,
y_test_bin, vectorizer, base_classifier, experiment_name,
labels_list):
    """Runs a single multi-label classification experiment and
returns results."""
    print(f"\n--- 开始运行多标签实验: {experiment_name}
---")

    # Use OneVsRestClassifier to handle multi-label scenario
with a base classifier
    multilabel_classifier                                  =
OneVsRestClassifier(base_classifier, n_jobs=-1) # Use all CPU
cores

    # Create pipeline
    pipeline = Pipeline([
        ('vectorizer', vectorizer),
        ('classifier', multilabel_classifier)
    ])

    # Train
    print("开始训练模型...")
    try:
        pipeline.fit(X_train, y_train_bin)
        print("模型训练完成。")
    except Exception as e:
```

```
            print(f"错误：模型训练失败 for '{experiment_name}'. Error: {e}")
            return {'name': experiment_name, 'status': 'failed', 'error': str(e)}

    # Predict
    print("开始在测试集上预测...")
    try:
        y_pred_bin = pipeline.predict(X_test)
        print("预测完成。")
    except Exception as e:
        print(f"错误：模型预测失败 for '{experiment_name}'. Error: {e}")
        return {'name': experiment_name, 'status': 'failed', 'error': str(e), 'pipeline': pipeline}

    # Evaluate
    print(f"\n评估指标 ({experiment_name}):")
    try:
        subset_acc = multilabel_accuracy_score(y_test_bin, y_pred_bin)
        hamming = hamming_loss(y_test_bin, y_pred_bin)
        precision_micro = precision_score(y_test_bin, y_pred_bin, average='micro', zero_division=0)
        recall_micro = recall_score(y_test_bin, y_pred_bin, average='micro', zero_division=0)
        f1_micro = f1_score(y_test_bin, y_pred_bin, average='micro', zero_division=0)
        precision_macro = precision_score(y_test_bin, y_pred_bin, average='macro', zero_division=0)
        recall_macro = recall_score(y_test_bin, y_pred_bin, average='macro', zero_division=0)
        f1_macro = f1_score(y_test_bin, y_pred_bin, average='macro', zero_division=0)
        precision_weighted = precision_score(y_test_bin, y_pred_bin, average='weighted', zero_division=0)
        recall_weighted = recall_score(y_test_bin, y_pred_bin, average='weighted', zero_division=0)
        f1_weighted = f1_score(y_test_bin, y_pred_bin,
```

```
average='weighted', zero_division=0)

            print(f"子集准确率 (Exact Match Ratio): {subset_acc:.4f}")
            print(f"汉明损失 (Hamming Loss): {hamming:.4f} (越低越好)")
            print(f" Micro Avg Precision: {precision_micro:.4f}")
            print(f" Micro Avg Recall: {recall_micro:.4f}")
            print(f" Micro Avg F1-Score: {f1_micro:.4f}")
            print(f" Macro Avg Precision: {precision_macro:.4f}")
            print(f" Macro Avg Recall: {recall_macro:.4f}")
            print(f" Macro Avg F1-Score: {f1_macro:.4f}")
            print(f" Weighted Avg Precision: {precision_weighted:.4f}")
            print(f" Weighted Avg Recall: {recall_weighted:.4f}")
            print(f" Weighted Avg F1-Score: {f1_weighted:.4f}")

            # Optional: Print classification report summary
            # report = classification_report(y_test_bin, y_pred_bin, target_names=labels_list, zero_division=0)
            # report_lines = report.split('\n')
            # print("\n分类报告摘要:")
            # print('\n'.join(report_lines[-4:])) # Print micro, macro, weighted, samples avg

        except Exception as e:
            print(f"错误：计算评估指标时出错 for '{experiment_name}'. Error: {e}")
            return {'name': experiment_name, 'status': 'evaluation_error', 'error': str(e), 'pipeline': pipeline}
```

```
    # Store results
    results = {
        'name': experiment_name, 'status': 'success',
        'subset_accuracy':    subset_acc,    'hamming_loss':
hamming,
        'precision_micro':  precision_micro,  'recall_micro':
recall_micro, 'f1_micro': f1_micro,
        'precision_macro': precision_macro, 'recall_macro':
recall_macro, 'f1_macro': f1_macro,
        'precision_weighted':            precision_weighted,
'recall_weighted': recall_weighted, 'f1_weighted': f1_weighted,
        'y_true_binarized':  y_test_bin,  'y_pred_binarized':
y_pred_bin, # Optional: store predictions
        'labels_list': labels_list, 'pipeline': pipeline
    }
    print(f"--- 实验 {experiment_name} 完成 ---")
    return results

# --- Main Execution Block ---
if __name__ == "__main__":
    print("="*50)
    print(" 开始执行多标签文本分类实验 (法律文书) ")
    print("="*50)

    # 1. 加载数据
    print("\n 步骤 1: 加载数据...")
    df = load_data(DATA_FILE)
    if df is None or df.empty:
        print("\n 数据加载失败或为空，程序退出。")
        exit()

    # 2. 数据清洗
    print("\n 步骤 2: 数据清洗...")
    initial_rows = len(df)
    df.dropna(subset=['sentence'], inplace=True)  #  Remove
rows with missing sentence
    df = df[df['sentence'].str.strip() != '']   # Remove rows
```

```
with empty/whitespace sentence
    # Ensure labels column contains lists, default to empty list if not
    df['labels'] = df['labels'].apply(lambda x: x if isinstance(x, list) else [])
    rows_after_cleaning = len(df)
    print(f"原始数据 {initial_rows} 行，清洗后剩余 {rows_after_cleaning} 行。")
    if df.empty:
        print("错误：清洗后没有有效数据。程序退出。")
        exit()

    # 3. 加载停用词
    print("\n 步骤 3：加载停用词...")
    stopwords = load_stopwords(STOPWORDS_FILE)

    # 4. 预处理文本数据
    print("\n 步骤 4：文本预处理 (分词、去停用词)...")
    df['processed_text'] = df['sentence'].apply(lambda x: preprocess_text(x, stopwords))
    print("文本预处理完成。")
    # Remove rows where text became empty after processing
    rows_before_empty_check = len(df)
    df = df[df['processed_text'].str.strip() != '']
    rows_after_empty_check = len(df)
    if rows_after_empty_check < rows_before_empty_check:
        print(f"\n 移除了 {rows_before_empty_check - rows_after_empty_check} 行 (因预处理后文本为空)。")
    print(f"最终用于模型训练的数据集形状：{df.shape}")
    if df.empty:
        print("错误：预处理后没有有效数据。程序退出。")
```

```
            exit()

    # 5. 标签二值化

    print("\n 步骤 5: 标签二值化...")

    mlb = MultiLabelBinarizer()
    try:
        y_binarized = mlb.fit_transform(df['labels'])
        labels_list = mlb.classes_ # Get unique labels

        print(f"共发现 {len(labels_list)} 个唯一标签: 
{list(labels_list)}")

        print("标签二值化完成。二值化标签矩阵形状:", 
y_binarized.shape)
        # Add binarized lists to df for potential later use 
(like filtering for word clouds)
        df['binarized_labels_list'] = list(y_binarized)
    except Exception as e:

        print(f"标签二值化时出错: {e}")

        traceback.print_exc()
        exit()


    # 6. 划分数据集

    print("\n 步骤 6: 划分训练集和测试集...")

    X = df['processed_text']
    y = y_binarized
    if len(X) < 2:

        print("错误：数据太少，无法进行训练/测试划分。
")

        exit()

    try:
        X_train, X_test, y_train_binarized, y_test_binarized 
= train_test_split(
            X, y, test_size=0.25, random_state=42 # Use 
25% for testing, fixed random state
        )
```

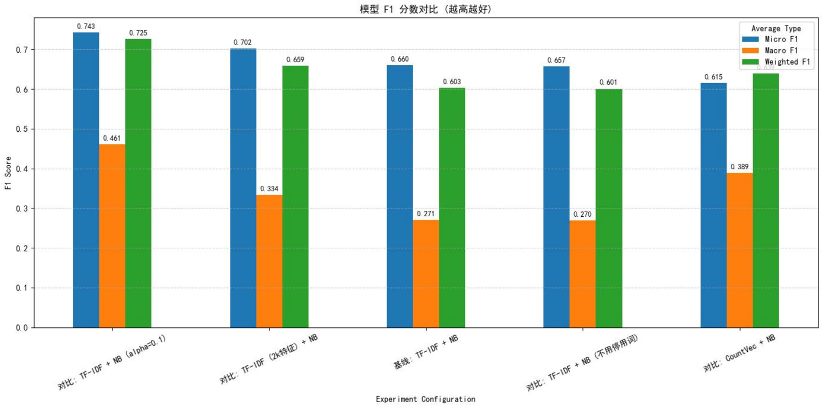
```
        print(f"数据集划分完成:")
        print(f"  训练集样本数量: {X_train.shape[0]}, 测试集样本数量: {X_test.shape[0]}")
        print(f"  训练集标签矩阵形状: {y_train_binarized.shape}, 测试集标签矩阵形状: {y_test_binarized.shape}")
    except Exception as e:
        print(f"划分数据集时出错: {e}")
        traceback.print_exc()
        exit()

    # --- 步骤 7: 定义和运行实验 ---
    print("\n步骤 7: 定义和运行分类实验...")
    results_list = []

    # --- Experiment Configurations ---
    base_nb = MultinomialNB(alpha=1.0)
    tfidf_vec = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
    count_vec = CountVectorizer(max_features=5000, ngram_range=(1, 2))

    # --- Experiment 1: Baseline (TF-IDF + NB) ---
    res1 = run_multilabel_experiment(X_train, y_train_binarized, X_test, y_test_binarized,
                                     tfidf_vec, base_nb, "基线: TF-IDF + NB", labels_list)
    if res1.get('status') == 'success': results_list.append(res1)

    # --- Experiment 2: Count Vectorizer + NB ---
    res2 = run_multilabel_experiment(X_train, y_train_binarized, X_test, y_test_binarized,
                                     count_vec, base_nb, "对比: CountVec + NB", labels_list)
```

```
    if res2.get('status') == 'success': results_list.append(res2)

    # --- Experiment 3: TF-IDF + NB (Lower Alpha) ---
    res3 = run_multilabel_experiment(X_train, y_train_binarized, X_test, y_test_binarized,
                                     tfidf_vec, MultinomialNB(alpha=0.1), # Change alpha
                                     "对比: TF-IDF + NB (alpha=0.1)", labels_list)
    if res3.get('status') == 'success': results_list.append(res3)

    # --- Experiment 4: TF-IDF (Fewer Features) + NB ---
    res4 = run_multilabel_experiment(X_train, y_train_binarized, X_test, y_test_binarized,
                                     TfidfVectorizer(max_features=2000, ngram_range=(1, 2)), # Fewer features
                                     base_nb, "对比: TF-IDF (2k 特征) + NB", labels_list)
    if res4.get('status') == 'success': results_list.append(res4)

    # --- Experiment 5: TF-IDF + NB (No Stopwords) ---
    print("\n为'无停用词'实验重新预处理训练/测试数据...")
    X_train_ns = X_train.apply(lambda x: preprocess_text(x, set())) # Apply preprocess without stopwords
    X_test_ns = X_test.apply(lambda x: preprocess_text(x, set()))
    # Filter out potential empty strings after reprocessing (unlikely but safe)
    train_mask = X_train_ns.str.strip() != ''
    test_mask = X_test_ns.str.strip() != ''
    X_train_ns_f = X_train_ns[train_mask]
    y_train_bin_ns_f = y_train_binarized[train_mask] # Filter labels accordingly
    X_test_ns_f = X_test_ns[test_mask]
    y_test_bin_ns_f = y_test_binarized[test_mask]    # Filter labels accordingly

    if not X_train_ns_f.empty and not X_test_ns_f.empty:
        res5 = run_multilabel_experiment(X_train_ns_f, y_train_bin_ns_f, X_test_ns_f, y_test_bin_ns_f,
```

```
tfidf_vec, base_nb, "对比：TF-IDF + NB (不用停用词)", labels_list)
            if res5.get('status') == 'success': results_list.append(res5)
        else:
            print("警告: 无停用词处理后训练或测试集为空，跳过此实验。")


    # --- 步骤 8: 结果对比与可视化 ---
    print("\n" + "="*20 + " 实验结果对比 " + "="*20)
    if not results_list:
        print("没有成功的实验结果可供对比。")
    else:
        # Create comparison DataFrame
        comparison_data = [{
            '实验名称': res['name'],
            '子集准确率': res.get('subset_accuracy', np.nan), # Use .get for safety
            '汉明损失': res.get('hamming_loss', np.nan),
            'Micro F1': res.get('f1_micro', np.nan),
            'Macro F1': res.get('f1_macro', np.nan),
            'Weighted F1': res.get('f1_weighted', np.nan)
        } for res in results_list]
        comparison_df = pd.DataFrame(comparison_data).sort_values(by='Micro F1', ascending=False).reset_index(drop=True)

        print("\n实验结果汇总 (按 Micro F1 降序):")
        pd.set_option('display.max_colwidth', 80) # Adjust column width
        pd.set_option('display.width', 120)        # Adjust total width
```

```
            print(comparison_df.round(4))                 #
Round to 4 decimal places

            # --- Plotting ---
            # Plot F1 Scores Comparison
            plot_metrics_f1 = ['Micro F1', 'Macro F1', 'Weighted
F1']
            comp_f1 = comparison_df.set_index('实验名称
')[plot_metrics_f1]
            ax_f1 = comp_f1.plot(kind='bar', figsize=(14, 7),
rot=25,
                                 title='模型 F1 分数
对比 (越高越好)')
            plt.ylabel('F1 Score')
            plt.xlabel('Experiment Configuration')
            plt.ylim(bottom=0)
            plt.legend(title="Average Type", loc='best')
            plt.grid(axis='y', linestyle='--', alpha=0.6)
            plt.tight_layout()
            for container in ax_f1.containers:
                ax_f1.bar_label(container,        fmt='%.3f',
padding=3, fontsize=9)
            plt.show()

            # Plot Hamming Loss Comparison
            comp_loss = comparison_df.set_index('实验名称')[['
汉明损失']]
            ax_loss = comp_loss.plot(kind='bar', figsize=(12, 6),
rot=25, color='tomato',
                                     title='模型汉明
损失对比 (越低越好)')
            plt.ylabel('Hamming Loss')
            plt.xlabel('Experiment Configuration')
            plt.ylim(bottom=0)
            plt.grid(axis='y', linestyle='--', alpha=0.6)
            plt.tight_layout()
            for container in ax_loss.containers:
```

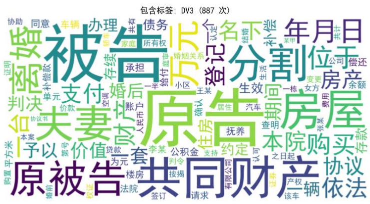
```
            ax_loss.bar_label(container,        fmt='%.3f',
padding=3, fontsize=9)
        plt.show()


    # --- 步骤 9: 其他可视化 (词云图) ---

    print("\n步骤 9: 生成词云图 (基于训练集)...")
    # Create a temporary DataFrame with training data needed
for plotting
    # Use the 'binarized_labels_list' column added earlier
    train_df_for_wc = df.loc[X_train.index].copy() # Select
training rows
    # Ensure the necessary columns exist
    if 'processed_text' in train_df_for_wc.columns and
'binarized_labels_list' in train_df_for_wc.columns:
         plot_word_clouds_multilabel(train_df_for_wc,

label_col='binarized_labels_list',

text_col='processed_text',

label_names=labels_list,

num_labels_to_plot=8) # Plot for top 8 labels
    else:
        print("错误：无法生成词云图，训练数据 DataFrame
缺少必要列。")



    print("\n" + "="*50)
    print(" 所有实验及可视化已完成 ")
    print("="*50)
```



模型 F1 分数对比（越高越好）
Average Type
Micro F1
Macro F1
Weighted F1
F1 Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.743
0.461
0.725
0.702
0.334
0.659
0.660
0.271
0.603
0.657
0.270
0.601
0.615
0.389
对比：TF-IDF + NB (alpha=0.1)
对比：TF-IDF (2k特征) + NB
基线：TF-IDF + NB
对比：TF-IDF + NB (不用停用词)
对比：CountVec + NB
Experiment Configuration

包含标签：DV1 (1833 次)
被告
原告
抚养
年月日
抚养费
离婚
生活
孩子

包含标签：DV2 (1121 次)
原告
被告
抚养费
抚养
离婚
生活



包含标签：DV3 (887 次)
原告
被告
共同财产
原被告
夫妻
房屋
分割

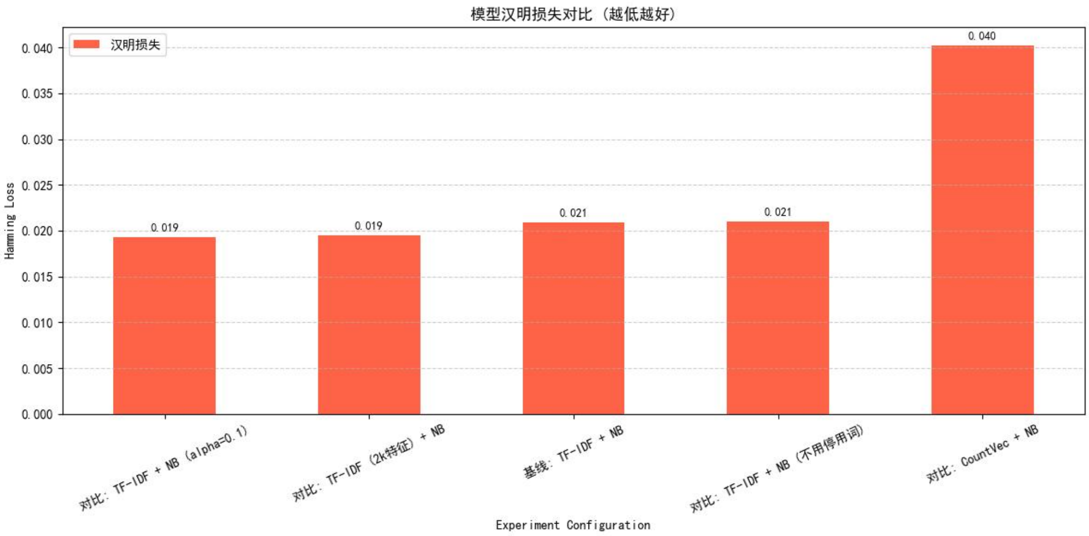
包含标签：DV4 (446 次)
抚养费
原告
被告
每月
支付
抚养

包含标签：DV5 (369 次)
原告
被告
房屋
共同财产
位于
夫妻

包含标签：DV6 (288 次)
原告
被告
分居
原被告
离婚
生活

包含标签：DV8 (286 次)
原告
被告
抚养费
支付
抚养
每月

包含标签：DV7 (280 次)
离婚
原告
被告
起诉
法院
年月日




```
数据样本（前5行）:
   labels                                           sentence
0      []     原告林某某诉称：我与被告经人介绍建立恋爱关系，于1995年在菏泽市民政局办理结婚登记手续。
1   [DV1]  1998年2月份生一女李某乙，2005年4月15日生次女李某丙，2007年11月生一女李某丁。
2      []                    由于婚前缺乏了解、草率结婚，婚后发现双方性格不合、经常生气吵架。
3  [DV13]                                            被告对家庭不管不问。
4   [DV6]                                   因感情不和，我与被告从2013年分居至今。
```

```
成功设置 Matplotlib 字体为 SimHei。 WordCloud 字体路径: C:\Windows\Fonts\simhei.ttf
============================================================
 开始执行多标签文本分类实验（法律文书）
============================================================

步骤 1: 加载数据...
Attempting to load data from: D:\PaddlePaddle-EfficientNetV2\PaddleClas-EfficientNet\离婚诉讼文本.json
尝试将整个文件作为单个 JSON 列表加载...
将整个文件作为 JSON 列表加载失败: Extra data: line 2 column 1 (char 1601)
将尝试逐行解析 JSON...

文件读取和解析完成。
总共成功加载并验证了 11685 条记录。

成功创建 DataFrame。
数据集形状: (11685, 2)
```

```
独立标签出现次数统计:
DV1     2402
DV2     1469
DV3     1212
DV4      585
DV5      515
DV7      381
DV6      378
DV8      364
DV9      361
DV10     317
DV12     173
DV13     160
DV11     109
DV19      73
DV17      63
DV16      62
DV18      59
DV20      57
DV15      56
DV14      54
Name: count, dtype: int64
```

```
句子标签数量分布:
label_count
0    6589
1    2665
2    1589
3     422
4     372
5      37
6       9
7       2
Name: count, dtype: int64

步骤 2: 数据清洗...
原始数据 11685 行，清洗后剩余 11685 行。

步骤 3: 加载停用词...
从 停用词表.txt 加载了 1627 个停用词。

步骤 4: 文本预处理 (分词、去停用词)...
Building prefix dict from the default dictionary ...
Dumping model to file cache C:\Users\86198\AppData\Local\Temp\jieba.cache
Loading model cost 0.546 seconds.
Prefix dict has been built successfully.
文本预处理完成。
```

```
移除了 20 行 (因预处理后文本为空)。
最终用于模型训练的数据集形状: (11665, 3)

步骤 5: 标签二值化...
共发现 20 个唯一标签: ['DV1', 'DV10', 'DV11', 'DV12', 'DV13', 'DV14', 'DV15', 'DV16', 'DV17', 'DV18', 'DV19', 'DV2', 'DV20', 'DV3', 'DV4', 'DV5', 'DV6'
标签二值化完成。二值化标签矩阵形状: (11665, 20)

步骤 6: 划分训练集和测试集...
数据集划分完成:
  训练集样本数量: 8748, 测试集样本数量: 2917
  训练集标签矩阵形状: (8748, 20), 测试集标签矩阵形状: (2917, 20)

步骤 7: 定义和运行分类实验...

--- 开始运行多标签实验: 基线: TF-IDF + NB ---
开始训练模型...
模型训练完成。
开始在测试集上预测...
预测完成。
```

```
评估指标 (基线: TF-IDF + NB):
  子集准确率 (Exact Match Ratio): 0.7041
  汉明损失 (Hamming Loss):     0.0209 (越低越好)
  Micro Avg Precision:          0.8415
  Micro Avg Recall:             0.5423
  Micro Avg F1-Score:           0.6596
  Macro Avg Precision:          0.4856
  Macro Avg Recall:             0.2230
  Macro Avg F1-Score:           0.2711
  Weighted Avg Precision:       0.7827
  Weighted Avg Recall:          0.5423
  Weighted Avg F1-Score:        0.6031
--- 实验 基线: TF-IDF + NB 完成 ---


--- 开始运行多标签实验: 对比: CountVec + NB ---
开始训练模型...
模型训练完成。
开始在测试集上预测...
预测完成。
```

```
评估指标 (对比: CountVec + NB):
  子集准确率 (Exact Match Ratio): 0.5166
  汉明损失 (Hamming Loss):     0.0402 (越低越好)
  Micro Avg Precision:         0.4779
  Micro Avg Recall:            0.8634
  Micro Avg F1-Score:          0.6152
  Macro Avg Precision:         0.2964
  Macro Avg Recall:            0.5988
  Macro Avg F1-Score:          0.3889
  Weighted Avg Precision:      0.5209
  Weighted Avg Recall:         0.8634
  Weighted Avg F1-Score:       0.6393
--- 实验 对比: CountVec + NB 完成 ---

--- 开始运行多标签实验: 对比: TF-IDF + NB (alpha=0.1) ---
开始训练模型...
模型训练完成。
开始在测试集上预测...
预测完成。
```

```
==================== 实验结果对比 ====================

实验结果汇总 (按 Micro F1 降序):
                          实验名称    子集准确率     汉明损失  Micro F1  Macro F1  Weighted F1
0  对比: TF-IDF + NB (alpha=0.1)  0.7189  0.0193    0.7428    0.4608       0.7253
1      对比: TF-IDF (2k特征) + NB  0.7158  0.0195    0.7022    0.3338       0.6586
2             基线: TF-IDF + NB  0.7041  0.0209    0.6596    0.2711       0.6031
3     对比: TF-IDF + NB (不用停用词)  0.7035  0.0210    0.6571    0.2696       0.6009
4           对比: CountVec + NB  0.5166  0.0402    0.6152    0.3889       0.6393

步骤 9: 生成词云图 (基于训练集)...

正在为频率最高的 8 个标签生成词云图...

==================================================
 所有实验及可视化已完成
==================================================

进程已结束，退出代码为 0
```

```
评估指标 (对比: TF-IDF + NB (不用停用词)):
  子集准确率 (Exact Match Ratio): 0.7035
  汉明损失 (Hamming Loss):     0.0210 (越低越好)
  Micro Avg Precision:         0.8413
  Micro Avg Recall:            0.5391
  Micro Avg F1-Score:          0.6571
  Macro Avg Precision:         0.4846
  Macro Avg Recall:            0.2215
  Macro Avg F1-Score:          0.2696
  Weighted Avg Precision:      0.7819
  Weighted Avg Recall:         0.5391
  Weighted Avg F1-Score:       0.6009
--- 实验 对比: TF-IDF + NB (不用停用词) 完成 ---
```

```
评估指标 (对比: TF-IDF (2k特征) + NB):
  子集准确率 (Exact Match Ratio): 0.7158
  汉明损失 (Hamming Loss):     0.0195 (越低越好)
  Micro Avg Precision:         0.8173
  Micro Avg Recall:            0.6155
  Micro Avg F1-Score:          0.7022
  Macro Avg Precision:         0.4706
  Macro Avg Recall:            0.2908
  Macro Avg F1-Score:          0.3338
  Weighted Avg Precision:      0.7573
  Weighted Avg Recall:         0.6155
  Weighted Avg F1-Score:       0.6586
--- 实验 对比: TF-IDF (2k特征) + NB 完成 ---

为'无停用词'实验重新预处理训练/测试数据...

--- 开始运行多标签实验: 对比: TF-IDF + NB (不用停用词) ---
开始训练模型...
模型训练完成。
开始在测试集上预测...
预测完成。
```

```
评估指标 (对比: TF-IDF + NB (alpha=0.1)):
  子集准确率 (Exact Match Ratio): 0.7189
  汉明损失 (Hamming Loss):     0.0193 (越低越好)
  Micro Avg Precision:         0.7391
  Micro Avg Recall:            0.7466
  Micro Avg F1-Score:          0.7428
  Macro Avg Precision:         0.5617
  Macro Avg Recall:            0.4536
  Macro Avg F1-Score:          0.4608
  Weighted Avg Precision:      0.7244
  Weighted Avg Recall:         0.7466
  Weighted Avg F1-Score:       0.7253
--- 实验 对比: TF-IDF + NB (alpha=0.1) 完成 ---

--- 开始运行多标签实验: 对比: TF-IDF (2k特征) + NB ---
开始训练模型...
模型训练完成。
开始在测试集上预测...
预测完成。
```

6.实验心得

本次实验旨在探索朴素贝叶斯分类器在法律裁判文书（离婚诉讼领域）多标签案情要素分类任务中的应用效果。实验基于约 1.17 万条带有案情要素标签的句子数据，采用 TF-IDF 和词频（CountVectorizer）作为文本表示方法，并对比了不同配置（如停用词使用、贝叶斯平滑参数 alpha、特征数量）对模型性能的影响。

主要发现与分析：

数据特点与预处理：

数据集包含 11685 条有效记录，共识别出 20 个不同的案情要素标签（DV1-DV20）。

标签分布呈现明显的不平衡性，如 'DV1'（子女相关）出现次数远超其他标签，而部分标签如 'DV14','DV15' 等出现次数较少。这为后续模型在稀有标签上的表现带来了挑战。

文本预处理（分词、去除非中文、去停用词）后，有少量（20 条）文本变为空，说明原始文本中可能存在无意义或非中文内容，预处理是必要的。

基线模型性能：

采用 TF-IDF (5k 特征, ngram 1-2) 和 MultinomialNB (alpha=1.0) 作为基线模型，在测试集上获得了 0.7041 的子集准确率和 0.6596 的 Micro F1 分数。

Micro F1 (0.66) 远高于 Macro F1 (0.27)，这进一步印证了数据标签不平衡对模型性能的影响——模型在样本量大的常见标签上表现尚可，但在稀有标签上的平均表现较差。

汉明损失 (Hamming Loss) 约为 0.0209，表示平均每次预测中，约有 2.1% 的标签被错误预测（预测为 1 而实际为 0，或反之）。

不同配置对比分析：

向量化方法： TF-IDF 在综合性能上优于词频（CountVectorizer）。虽然 CountVectorizer 获得了更高的召回率（Micro Recall 0.86 vs TF-IDF 0.54），但其精确率较低（Micro Precision 0.48 vs TF-IDF 0.84），导致 Micro F1 分数低于 TF-IDF (0.615 vs 0.660)。这表明 TF-IDF 的词频逆文档频率权重对于区分不同标签的文本特征更为有效。

平滑参数 (Alpha)： 调整朴素贝叶斯的平滑参数 alpha 从 1.0 降至 0.1 对模型性能提升最为显著。该配置（TF-IDF + NB alpha=0.1）在所有实验中取得了最佳效果，Micro F1 提升至 0.7428，Macro F1 提升至 0.4608，子集准确率和汉明损失也有所改善。这表明较小的 alpha 值（较少的平滑）更适合此数据集和模型。

特征数量： 将 TF-IDF 特征数从 5000 减少到 2000 时，相较于 基线 模型（alpha=1.0），性能有所提升（Micro F1 从 0.660 提升至 0.702），但仍不及调整 alpha=0.1 后的效果。这可能意味着 5000 个特征对于 alpha=1.0 的模型来说可能过多或包含噪声，但调整模型本身的参数(如 alpha) 是更有效的优化途径。

停用词： 在本实验中，使用提供的停用词表与不使用停用词表，对 TF-IDF + NB (alpha=1.0) 的模型性能影响微乎其微（Micro F1 几乎不变，其他指标也类似）。这可能是因为 TF-IDF 本身会降低常见词的权重，或者所用的停用词表对区分案情要素的关键信息影响不大。

可视化辅助： 实验包含了生成词云图的步骤，这有助于直观理解与高频案情要素（如 DV1, DV2, DV3 等）相关的核心词汇，为特征工程或结果解释提供定性参考。

结论与展望：

朴素贝叶斯结合 TF-IDF 是一种可行且计算高效的法律文本多标签分类基线方法。

超参数调优（如朴素贝叶斯的 alpha）对模型性能至关重要，效果优于单纯调整特征数量。在本实验中，alpha=0.1 的配置表现最佳。

数据集的标签不平衡是影响模型（尤其是 Macro 平均指标）的关键因素。

标准的通用停用词表在此任务中效果有限，未来可考虑构建法律领域专用的停用词表。

未来改进方向：

尝试更复杂的模型，如 SVM、Logistic Regression 或基于深度学习的模型（如 BERT、LSTM）可能获得更好的性能。

采用针对多标签不平衡问题的策略，如标签幂集转换、分类器链、欠采样/过采样技术（需小心应用于多标签）、代价敏感学习等。

进行更细致的特征工程，例如引入词性、句法结构、法律实体识别等特征。

对模型进行更详细的错误分析，了解在哪些标签和哪些类型的句子上表现不佳，以便针对性改进。