华侨大学计算机科学与技术学院

《信息内容安全实验》实验报告

班级	信息安全 1 班	学号	2325181017	姓名	廖媛
实验日期	2026.5.27	实验项目	文本分类实验		

一、实验目的

1. 掌握朴素贝叶斯分类器的原理

2. 了解 sklearn 框架，掌握使用 sklearn 进行文本分类的基本方法

二、实验内容

使用 20 newsgroups 数据集，分别采用以下三种方法进行文本分类：

1. TF-IDF + MultinomialNB（教程基准方法）

2. TF-IDF + SVM（线性核）

3. CountVectorizer（原始词频）+ MultinomialNB（变体方法）

最后对比三种方法的分类效果。

任务 1：环境搭建与环境加载

```python
# 导入所需库
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 设置中文显示和绘图风格
plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei', 'DejaVu Sans']
plt.rcParams['axes.unicode_minus'] = False

print('所有库导入成功！')
```

Python

```python
# 选择4个类别（与教程一致）
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']

# 加载训练集
twenty_train = fetch_20newsgroups(
    subset='train',
    categories=categories,
    shuffle=True,
    random_state=42
)

# 加载测试集
twenty_test = fetch_20newsgroups(
    subset='test',
    categories=categories,
    shuffle=True,
    random_state=42
)

print(f'训练集样本数: {len(twenty_train.data)}')
print(f'测试集样本数: {len(twenty_test.data)}')
print(f'类别: {twenty_train.target_names}')
print(f'总类别数: {len(twenty_train.target_names)}')
```

Python

```python
# 查看数据分布
train_counts = pd.Series(twenty_train.target).value_counts().sort_index()
test_counts = pd.Series(twenty_test.target).value_counts().sort_index()

dist_df = pd.DataFrame({
    '类别': [twenty_train.target_names[i] for i in range(len(categories))],
    '训练集': train_counts.values,
    '测试集': test_counts.values
})
dist_df
```

Python

```python
# 查看一条样本
print('=== 第一条训练样本 ===')
print(f'类别: {twenty_train.target_names[twenty_train.target[0]]}')
print(f'内容（前500字符）:\n{twenty_train.data[0][:500]}')
```

Python

任务 2：TF-IDF+MutinomialNB 分类器

```python
# 使用 TF-IDF 进行特征提取
tfidf_vectorizer = TfidfVectorizer(
    stop_words='english',     # 去除英文停用词
    max_df=0.5,               # 忽略出现在超过50%文档中的词
    min_df=2,                 # 忽略出现在少于2个文档中的词
    max_features=None,
)

# 拟合并转换训练集
X_train_tfidf = tfidf_vectorizer.fit_transform(twenty_train.data)
# 转换测试集（仅transform，不拟合）
X_test_tfidf = tfidf_vectorizer.transform(twenty_test.data)

print(f'TF-IDF 特征矩阵形状 - 训练集: {X_train_tfidf.shape}')
print(f'TF-IDF 特征矩阵形状 - 测试集: {X_test_tfidf.shape}')
print(f'词汇表大小: {len(tfidf_vectorizer.vocabulary_)}')
```

Python

```python
# 训练 MultinomialNB 分类器
nb_tfidf = MultinomialNB(alpha=1.0)  # alpha=1.0 为 Laplace 平滑
nb_tfidf.fit(X_train_tfidf, twenty_train.target)

# 预测
y_pred_nb_tfidf = nb_tfidf.predict(X_test_tfidf)

# 评估
accuracy_nb_tfidf = accuracy_score(twenty_test.target, y_pred_nb_tfidf)
print('=' * 60)
print('【方法一】TF-IDF + MultinomialNB 分类结果')
print('=' * 60)
print(f'准确率 (Accuracy): {accuracy_nb_tfidf:.4f} ({accuracy_nb_tfidf*100:.2f}%)')
print()
print('分类报告:')
print(classification_report(
    twenty_test.target,
    y_pred_nb_tfidf,
    target_names=twenty_train.target_names
))
```

Python

任务 3：TF-IDF+SVM 分类器

```python
# 训练 SVM 分类器（线性核），复用 TF-IDF 特征
svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train_tfidf, twenty_train.target)

# 预测
y_pred_svm = svm.predict(X_test_tfidf)

# 评估
accuracy_svm = accuracy_score(twenty_test.target, y_pred_svm)
print('=' * 60)
print('【方法二】TF-IDF + SVM（线性核）分类结果')
print('=' * 60)
print(f'准确率 (Accuracy): {accuracy_svm:.4f} ({accuracy_svm*100:.2f}%)')
print()
print('分类报告:')
print(classification_report(
    twenty_test.target,
    y_pred_svm,
    target_names=twenty_train.target_names
))
```

Python

任务 4:CountVectorize + MutinomialNB 分类器（变体方式）

关键区别：这里不使用 TF-IDF 加权，直接使用原始词频（counts）作为特征。

```python
# 使用 CountVectorizer 进行特征提取（原始词频，不用 TF-IDF）
count_vectorizer = CountVectorizer(
    stop_words='english',     # 与 TF-IDF 保持一致的预处理
    max_df=0.5,
    min_df=2,
)

# 拟合并转换
X_train_counts = count_vectorizer.fit_transform(twenty_train.data)
X_test_counts = count_vectorizer.transform(twenty_test.data)

print(f'Count 特征矩阵形状 - 训练集: {X_train_counts.shape}')
print(f'Count 特征矩阵形状 - 测试集: {X_test_counts.shape}')
print(f'词汇表大小: {len(count_vectorizer.vocabulary_)}')
```

Python

```python
# 训练 MultinomialNB 分类器（使用原始词频）
nb_counts = MultinomialNB(alpha=1.0)
nb_counts.fit(X_train_counts, twenty_train.target)

# 预测
y_pred_nb_counts = nb_counts.predict(X_test_counts)

# 评估
accuracy_nb_counts = accuracy_score(twenty_test.target, y_pred_nb_counts)
print('=' * 60)
print('【方法三】CountVectorizer（原始词频）+ MultinomialNB 分类结果')
print('=' * 60)
print(f'准确率 (Accuracy): {accuracy_nb_counts:.4f} ({accuracy_nb_counts*100:.2f}%)')
print()
print('分类报告:')
print(classification_report(
    twenty_test.target,
    y_pred_nb_counts,
    target_names=twenty_train.target_names
))
```

Python

任务 4:三种方法结果对比与分析

```python
# 汇总三种方法的准确率
results = pd.DataFrame({
    '方法': [
        'TF-IDF + MultinomialNB',
        'TF-IDF + SVM（线性核）',
        'CountVectorizer + MultinomialNB'
    ],
    '准确率 (%)': [
        round(accuracy_nb_tfidf * 100, 2),
        round(accuracy_svm * 100, 2),
        round(accuracy_nb_counts * 100, 2)
    ]
})
results
```

Python

```python
# 绘制准确率对比柱状图
fig, ax = plt.subplots(figsize=(10, 5))

methods = results['方法'].values
accuracies = results['准确率 (%)'].values
colors = ['#3498db', '#2ecc71', '#e74c3c']

bars = ax.bar(methods, accuracies, color=colors, edgecolor='black', linewidth=1.2)

# 在柱子上标注数值
for bar, acc in zip(bars, accuracies):
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.5,
            f'{acc:.2f}%', ha='center', va='bottom', fontsize=13, fontweight='bold')

ax.set_ylabel('准确率 (%)', fontsize=12)
ax.set_title('三种文本分类方法准确率对比', fontsize=15, fontweight='bold')
ax.set_ylim(0, max(accuracies) + 10)
ax.grid(axis='y', alpha=0.3)

plt.tight_layout()
plt.savefig('accuracy_comparison.png', dpi=150, bbox_inches='tight')
plt.show()
print('对比图已保存为 accuracy_comparison.png')
```

Python

```python
# 提取各方法的每类 F1 分数用于对比
from sklearn.metrics import f1_score

f1_nb_tfidf = f1_score(twenty_test.target, y_pred_nb_tfidf, average=None)
f1_svm = f1_score(twenty_test.target, y_pred_svm, average=None)
f1_nb_counts = f1_score(twenty_test.target, y_pred_nb_counts, average=None)

f1_comparison = pd.DataFrame({
    '类别': twenty_train.target_names,
    'TF-IDF+NB': [round(f, 4) for f in f1_nb_tfidf],
    'TF-IDF+SVM': [round(f, 4) for f in f1_svm],
    'Counts+NB': [round(f, 4) for f in f1_nb_counts],
})

# 添加宏观平均
f1_comparison.loc[4] = [
    '宏观平均 (Macro Avg)',
    round(f1_nb_tfidf.mean(), 4),
    round(f1_svm.mean(), 4),
    round(f1_nb_counts.mean(), 4)
]

f1_comparison
```

Python

```python
# 绘制每类 F1 对比图
fig, ax = plt.subplots(figsize=(10, 5))

x = np.arange(len(categories))
width = 0.25

bars1 = ax.bar(x - width, f1_nb_tfidf, width, label='TF-IDF + NB', color='#3498db', edgecolor='black')
bars2 = ax.bar(x, f1_svm, width, label='TF-IDF + SVM', color='#2ecc71', edgecolor='black')
bars3 = ax.bar(x + width, f1_nb_counts, width, label='Counts + NB', color='#e74c3c', edgecolor='black')

ax.set_xlabel('类别', fontsize=12)
ax.set_ylabel('F1 分数', fontsize=12)
ax.set_title('三种方法在各类别上的 F1 分数对比', fontsize=14, fontweight='bold')
ax.set_xticks(x)
ax.set_xticklabels(categories, rotation=15, ha='right', fontsize=10)
ax.legend(fontsize=11)
ax.set_ylim(0, 1.05)
ax.grid(axis='y', alpha=0.3)

plt.tight_layout()
plt.savefig('f1_comparison.png', dpi=150, bbox_inches='tight')
plt.show()
print('F1对比图已保存为 f1_comparison.png')
```

Python

三、实验总结

1. 实验分析与结论

1）三种方法性能排序

根据实验结果，三种方法的准确率从高到低为：

排名	方法	准确率
1	CountVectorizer + MultinomialNB	93.81%

2	TFIDF + SVM（线性核）	92.28%
3	TFIDF + MultinomialNB	89.81%

2）方法一：TFIDF + MultinomialNB（准确率 89.81%）

这是教程的基准方法。TFIDF 通过逆文档频率（IDF）降低常见词的权重，突出具有区分度的稀有词。MultinomialNB 在 TFIDF 特征上表现良好，但受限于其"特征条件独立"假设，在文本分类中天然弱于判别模型。

3）方法二：TFIDF + SVM（准确率 92.28%）

SVM 作为判别模型，通过最大化分类间隔找到最优决策边界，在高维稀疏文本特征空间中表现优异。相比 MultinomialNB，SVM 不依赖特征独立假设，因此能够捕捉词语之间的关联信息。实验结果也验证了 SVM 优于同特征下的 MultinomialNB。

4）方法三：CountVectorizer + MultinomialNB（准确率 93.81%）———— 本实验最佳

这是本实验最意外的发现。 使用原始词频（不做 TFIDF 加权）的 MultinomialNB 取得了最高的准确率（93.81%），甚至超过了 SVM。分析原因如下：

MultinomialNB 天然适配计数数据：MultinomialNB 基于多项分布建模，其概率估计直接使用词频计数。原始计数完美保留了这种分布特性，而 TFIDF 的归一化和 IDF 加权反而可能扭曲了词语在各类别中的条件概率估计。

停用词过滤已去除噪声：`stop_words='english'` 和 `max_df=0.5`、`min_df=2` 已经有效过滤了大量无信息词，使得原始计数中的"高频噪声"问题得到缓解。

IDF 可能过度惩罚：在 4 个语义差异较大的类别中，某些在多类中出现的词可能仍具有区分度（如 "god" 同时在 atheism 和 christian 中出现但用法不同），

IDF 会错误降低这些词的权重。

5）各方法在类别上的表现差异

从分类报告可以看出：

alt.atheism 是三个方法中 F1 最低的类别（0.85~0.91），可能是因为该类别讨论的话题（无神论）与其他类别有语义交叉。

soc.religion.christian 在 TFIDF + NB 下召回率高达 97%，但精确率只有 79%，说明该方法容易将其他类别误分为此类。

CountVectorizer + NB 在所有四个类别上都保持了 91% 以上的 F1 分数，表现出最均衡的分类能力。

2. 总结

本实验通过三种不同的特征表示和分类器组合，对 20 newsgroups 数据集进行文本分类，得到以下结论：

1. 特征表示方式影响显著：对于 MultinomialNB，原始词频反而优于 TFIDF（93.81% vs 89.81%），说明特征处理方式需要与分类器的数学特性相匹配。

2. SVM 稳健且强大：SVM 在文本分类中表现出色（92.28%），且不依赖特定的特征缩放方式。

3. 没有"一刀切"的最佳方案：本实验中 CountVectorizer + NB 表现最佳，但这受到数据集特性、类别数量和预处理参数的影响。在实际应用中应通过交叉验证选择最适合特定任务的组合。