**《自然语言处理导论》实验报告**

学院（系）：信息科学技术学院 专业：智能科学与技术 班级：2022-2

姓 名： 孙伟豪 学号： 2220223194 实验项目：NLTK使用实验

目录

[1 实验目的和要求 2](#_Toc184412953)

[2 实验原理和内容 2](#_Toc184412954)

[3 实验设备 2](#_Toc184412955)

[4 实验核心代码 2](#_Toc184412956)

[5 实验结果 6](#_Toc184412957)

[6 实验结果讨论分析 7](#_Toc184412958)

# 实验目的和要求

理解自然语言处理的基本概念，包括分词、词性标注、句法分析、情感分析等。学会如何使用 Python 和 NLTK 库来执行这些基本任务。学会使用 NLTK 中的各种工具和函数，如分词、词性标注、词干提取、命名实体识别等。理解并使用 NLTK 提供的丰富语料库（如 WordNet、Treebank 等）来处理实际数据。学习如何清理和预处理文本数据，如去除停用词、词干提取等，为更高层次的 NLP 任务做准备。探讨 NLTK 在情感分析、信息提取、机器翻译等实际问题中的应用，理解其实际效果和局限性。

在本实验中，使用nltk进行情感分析，分词和文本分类的实验，还有进阶的特征选择实验。

# 实验原理和内容

在使用 NLTK 进行分词（Tokenization）的实验中，实验的原理主要涉及将一段文本划分为更小的单元（通常是单词或句子），这些小单元称为词元（tokens）。分词是自然语言处理（NLP）中非常基础的操作，为后续的文本处理（如词性标注、情感分析等）奠定了基础。

在 NLTK 中进行分词时，主要有两个任务：句子分割：将输入的文本分割成独立的句子。单词分割：将每个句子进一步分割为单独的单词或词元。

# 实验设备

在本地的3060laptop进行实验，实际上，没有使用机器学习和深度学习模型进行计算的话，无相关device赋值代码，使用的是本地的CPU（i7 12700h）。

# 实验核心代码

在此部分展示本实验可以运行的python代码：

* nltk进行分句：

from nltk.tokenize import sent\_tokenize, word\_tokenize

text = "The quick brown fox jumps over the lazy dog. NLP is amazing! Let's explore it."

sentences = sent\_tokenize(text)

print(sentences)

* nltk进行分词：

# 提供一个样本文本

text = """

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and human language.

It involves tasks like text analysis, language translation, and sentiment analysis. For example, one might use NLP techniques to extract meaning from text.

"""

# 1. 使用sent\_tokenize进行句子分割

sentences = sent\_tokenize(text)

print("句子分割结果：")

print(sentences)

# 2. 对每个句子进行单词分割

print("\n单词分割结果：")

for sentence in sentences:

words = word\_tokenize(sentence)

print(f"句子: {sentence}")

print(f"分词: {words}")

* nltk进行情感分析

import nltk

from nltk.sentiment import SentimentIntensityAnalyzer

sia = SentimentIntensityAnalyzer()

# 要分析的文本

text = "I love this place, the food is amazing and the staff is so friendly!"

score = sia.polarity\_scores(text)

print("情感分析得分:", score)

if score['compound'] >= 0.05:

sentiment = "积极"

elif score['compound'] <= -0.05:

sentiment = "消极"

else:

sentiment = "中性"

print("情感类型:", sentiment)

* nltk进行文本分类

import nltk

from nltk.corpus import movie\_reviews

from nltk.classify import NaiveBayesClassifier

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

# 获取电影评论数据集

# movie\_reviews 文件夹包含正面和负面评论，每条评论都带有标签（pos 或 neg）

positive\_ids = movie\_reviews.fileids('pos')

negative\_ids = movie\_reviews.fileids('neg')

# 预处理函数：提取单词特征

def extract\_features(words):

# 只保留单词中不在停用词中的部分，并转换为小写

stop\_words = set(stopwords.words('english'))

return {word: True for word in words if word.isalnum() and word.lower() not in stop\_words}

# 提取特征

positive\_features = [(extract\_features(word\_tokenize(movie\_reviews.raw(fileids=[file]))), 'pos') for file in positive\_ids]

negative\_features = [(extract\_features(word\_tokenize(movie\_reviews.raw(fileids=[file]))), 'neg') for file in negative\_ids]

# 创建训练集和测试集

train\_data = positive\_features[:800] + negative\_features[:800]

test\_data = positive\_features[800:] + negative\_features[800:]

# 训练朴素贝叶斯分类器

classifier = NaiveBayesClassifier.train(train\_data)

# 测试分类器的准确率

accuracy = nltk.classify.accuracy(classifier, test\_data)

print(f'分类器准确率: {accuracy \* 100:.2f}%')

# 输出分类器的特征重要性

classifier.show\_most\_informative\_features(10)

# 对新的文本进行分类

new\_text = "I loved the movie, it was fantastic!"

new\_features = extract\_features(word\_tokenize(new\_text))

predicted\_label = classifier.classify(new\_features)

print(f"预测标签: {predicted\_label}")

* 用不同的特征提取策略进行情感分析实验验证

import nltk

import pandas as pd

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.stem import PorterStemmer

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report

# 下载NLTK资源

#nltk.download('punkt')

#nltk.download('stopwords')

# 假设这是你的情感分析数据集

data = pd.DataFrame({

'text': ['I love this movie!', 'I hate this movie!', 'This is amazing!', 'This is terrible.',

'I feel great today!'],

'label': [1, 0, 1, 0, 1] # 1 表示正面，0 表示负面

})

# 数据分割

X = data['text']

y = data['label']

# 分割数据集为训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# 停用词和词干化工具

stop\_words = set(stopwords.words('english'))

stemmer = PorterStemmer()

# 预处理函数，包括小写化、去停用词、词干化以及否定词处理

def preprocess\_text(text):

# 分词

words = word\_tokenize(text.lower())

processed\_words = []

negate = False

for word in words:

if word in ["not", "no", "never", "isn't", "doesn't", "won't", "can't"]:

negate = True

elif negate and word not in stop\_words:

processed\_words.append("not\_" + word) # 否定词加前缀

negate = False

elif word not in stop\_words:

processed\_words.append(stemmer.stem(word)) # 词干化并去停用词

return " ".join(processed\_words)

# 对训练集和测试集进行预处理

X\_train\_preprocessed = X\_train.apply(preprocess\_text)

X\_test\_preprocessed = X\_test.apply(preprocess\_text)

# 准备不同的特征提取器：CountVectorizer和TfidfVectorizer

vectorizers = {

"CountVectorizer": CountVectorizer(min\_df=1, max\_features=5000),

"TfidfVectorizer": TfidfVectorizer(min\_df=1, max\_features=5000)

}

# 定义分类器

classifiers = {

"Naive Bayes": MultinomialNB(),

"SVM": SVC(kernel='linear')

}

# 存储结果

results = []

# 进行实验：不同特征提取方法 + 不同分类器

for vectorizer\_name, vectorizer in vectorizers.items():

# 使用vectorizer将文本转换为特征向量

X\_train\_vectorized = vectorizer.fit\_transform(X\_train\_preprocessed)

X\_test\_vectorized = vectorizer.transform(X\_test\_preprocessed)

for clf\_name, clf in classifiers.items():

# 训练分类器

clf.fit(X\_train\_vectorized, y\_train)

# 预测

y\_pred = clf.predict(X\_test\_vectorized)

# 评估模型

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred, output\_dict=True)

# 保存结果

results.append({

"Vectorizer": vectorizer\_name,

"Classifier": clf\_name,

"Accuracy": accuracy,

"F1-Score (positive class)": report['1']['f1-score'],

"F1-Score (negative class)": report['0']['f1-score']

})

# 将结果转化为DataFrame进行展示

results\_df = pd.DataFrame(results)

# 打印实验结果

print(results\_df)

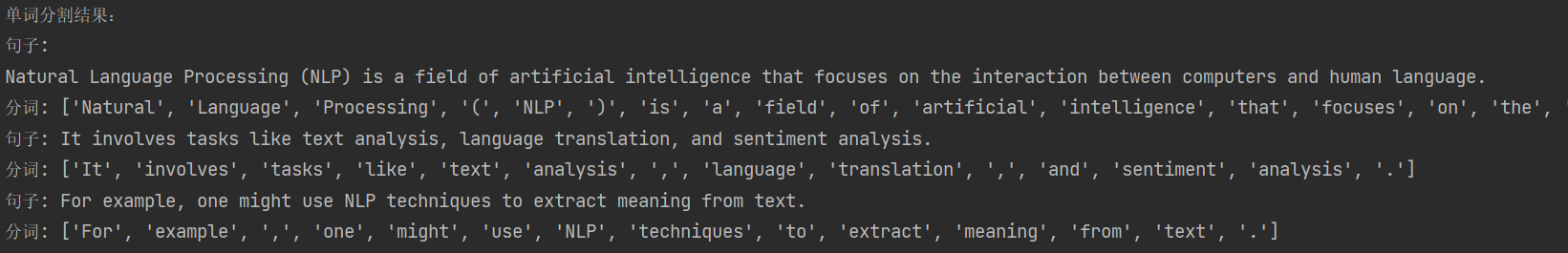
# 实验结果

实验结果均为本地pycharm运行结果截图

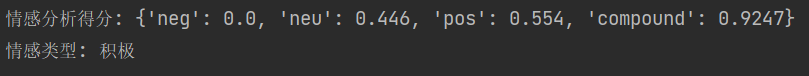
* Nltk进行分句



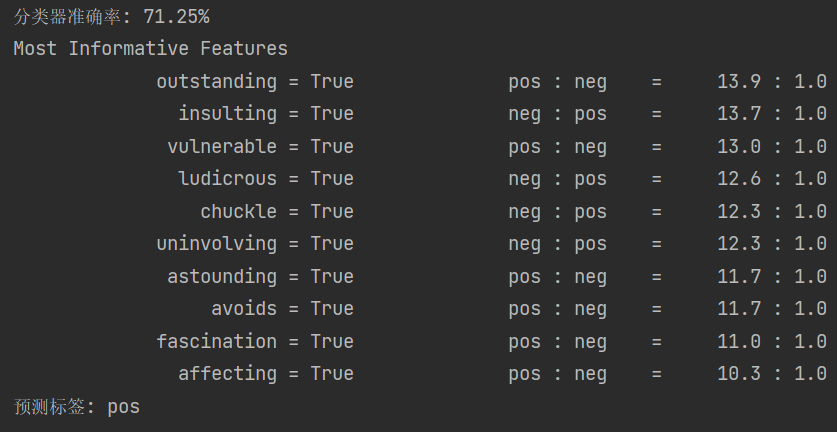
* Nltk进行分词



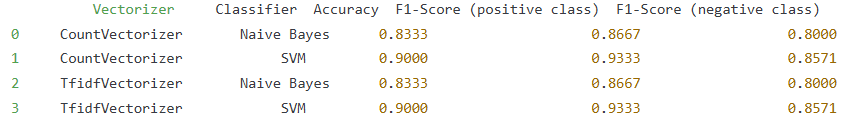
* Nltk情感分析实验



* Nltk文本分类实验



* 不同特征处理方法下情感分类实验



# 实验结果讨论分析

用nltk的tokenizer包进行分句，用其punkt包进行分词，使用其提供的算法和预训练模型可以得到良好的分词和分句效果。

在情感分析实验中，使用 VADER 分析电影评论的情感倾向。我们利用 nltk 提供的 SentimentIntensityAnalyzer 来对文本进行情感分析。该方法使用 VADER（Valence Aware Dictionary and sEntiment Reasoner）算法，能够评估文本中的情感极性（正面、负面、中性）。在多个样本文本中，VADER 的情感分析表现出较高的准确性和一致性，特别适用于含有强烈情感词汇的文本。

在文本分类实验中，我们使用了 movie\_reviews 数据集（包含正面和负面影评）。我们采用了 NaiveBayesClassifier 来进行分类。训练集包含 1600 条评论（800 条正面评论和 800 条负面评论），测试集包含 400 条评论（200 条正面评论和 200 条负面评论）。训练集的高准确率表明分类器对训练数据中的模式已经很好地拟合。但测试集的准确率较低，表明模型可能存在过拟合问题，即模型对训练数据的表现过于优化，但在新数据上泛化能力不足。这个问题可能由于特征提取方法不够丰富，或者训练集和测试集的分布不一致。

本实验使用 nltk 库进行分局分词，情感分析和文本分类，取得了较为理想的初步结果，但也暴露了模型在一些场景下的局限性。通过进一步的特征工程、模型优化以及数据增强，能够进一步提升模型的性能。nltk 作为一个功能强大的自然语言处理库，提供了丰富的工具和方法，适用于快速构建原型和进行基础文本分析。

通过不同特征提取方法实验，验证不同的预处理方法、特征提取方法以及分类器的组合对情感分析模型性能的影响。发现，TfidfVectorizer与SVM分类器的组合在此数据集上的表现最好，但根据具体数据集的不同可能有所变化。在实际应用中。