第7章 金融市场情绪分析

金融市场情绪分析是一种通过研究和评估市场参与者的情感和情绪，以预测或解释金融市场行为的方法。市场情绪可以对股票、债券、外汇、商品和其他金融资产的价格和波动产生重要影响，会影响金融市场的走势。在本章的内容中，将详细讲解使用深度学习技术实现金融市场情绪分析的知识，并通过具体实例来讲解各个知识点的用法。

7.1 情绪分析的概念与方法

情绪分析，也称为情感分析，是一种自然语言处理（NLP）技术，旨在识别和理解文本或语音中包含的情感、情绪和情感倾向。情绪分析的目标是确定文本或语音中的情感是积极、消极还是中性，并可以进一步细化为更多的情感类别。

7.1.1 情绪分析的基本概念

以下是情绪分析的基本概念。

（1）情感分类：情感分析的主要任务之一是将文本或语音中的情感分为不同的类别。最常见的情感包括以下三类。

☑ 积极情感（positive emotion）：包括喜悦、满足、兴奋等积极的情感状态。

☑ 消极情感（negative emotion）：包括愤怒、沮丧、焦虑等消极的情感状态。

☑ 中性情感（neutral emotion）：表示文本或语音中没有明显的情感表达。

（2）情感极性：除了将情感分类为积极、消极和中性外，情感分析还可以测量情感的强度或极性。例如，积极情感可以是高兴和兴奋，消极情感可以是悲伤和愤怒，它们可以有不同的强度。

（3）情感词汇：情感分析依赖于情感词汇，这些词汇是与情感相关的单词或短语。情感词汇通常被用于确定文本中的情感表达。

（4）语境理解：情感分析不仅要考虑情感词汇，还要考虑它们在文本中的上下文。同样的词汇在不同的上下文中可能具有不同的情感含义。

（5）机器学习和深度学习：情感分析通常使用机器学习和深度学习技术来自动识别和分类情感。这包括使用已标记的训练数据来训练情感分类模型。

（6）应用领域：情感分析在各种应用领域中都有广泛的用途，包括社交媒体监测、产品评论分析、市场研究、情感智能助手、客户服务反馈分析等。

（7）情感分析工具和 API：有许多情感分析工具和 API 可供使用，它们可以帮助分析文本或语音的情感。这些工具通常使用预训练的模型和情感词汇库来进行情感分析。

情感分析的应用场景广泛，可以帮助组织更好地理解他们的用户或客户的情感和情感反馈，以做出更明智的决策和改进产品及服务。此外，情感分析也在监测舆情、社交媒体分析和自然语言处理领域发挥着关键作用。

7.1.2 金融市场情绪的重要性

金融市场情绪能够对市场行为和资产价格波动产生深远的影响，以下几个方面展示了金融市场情绪的重要性。

- ☑ 价格波动：市场情绪可以导致资产价格的剧烈波动。积极情绪可能会推高资产价格，而消极情绪可能会导致价格下跌。这种价格波动可能给投资者和交易者带来巨大的风险或机会。
- ☑ 投资者行为：金融市场情绪对投资者的决策和行为会产生深远影响。情绪可以影响投资者的风险偏好，决定他们是买入还是卖出资产。例如，恐慌情绪可能导致抛售，而乐观情绪可能推高市场。
- ☑ 泡沫和崩盘：市场情绪可以导致资产价格产生泡沫，即价格远高于其内在价值。当情绪逆转时，泡沫可能会破裂，导致市场崩溃。这种情况下，情绪的失衡可能导致系统性风险。
- ☑ 市场动态：情绪可以在短期内推动市场，远离基本面的价值。这意味着投资者需要密切关注市场情绪，以及它是否与基本面相一致，以更好地管理风险。
- ☑ 舆情影响：媒体、社交媒体和舆论可以迅速传播情绪。市场参与者可能会受到新闻报道、社交媒体评论和专家观点的影响，从而改变他们的投资决策。
- ☑ 政策和法规： 政府和监管机构也受到市场情绪的影响。情绪可能会推动政策和法规的变化，影响金融市场的运作和稳定性。
- ☑ 投资策略：一些投资策略和交易策略以市场情绪为基础。例如，一些投资者使用情感分析工具来识别市场情绪的变化，以辅助他们的交易决策。

因此，理解金融市场情绪对投资者、交易者、机构和政策制定者都至关重要。市场情绪不仅是市场行为的重要驱动因素，还可以提供关于市场走向和潜在风险的有用见解。然

而，情绪分析应谨慎进行，因为它可能受到噪声和误导信息的影响，而且市场情绪本身也容易波动。因此，应将它与其他分析方法和基本面分析相结合，以制定更全面和明智的投资决策。

7.1.3　情绪分析的方法

情绪分析是一种通过自然语言处理技术来识别和理解文本或语音中包含的情感、情绪和情感倾向的方法，常用的情感分析方法如下。

1. 基于规则的方法

- ☑ 情感词汇词典：构建包含情感词汇的词汇表，每个词汇都分配一个情感极性（例如，积极、消极或中性）。在分析文本时，计算文本中包含的情感词汇的数量和情感极性，以确定文本的情感。
- ☑ 规则引擎：制定一组规则和模式，以检测情感表达方式，如否定语、程度副词等，然后根据这些规则来评估文本的情感。

2. 机器学习方法

- ☑ 有监督学习：使用带有情感标签的训练数据来训练情感分类模型，如朴素贝叶斯、支持向量机、随机森林等。模型学习文本中的模式并用于情感分类。
- ☑ 无监督学习：使用无标签的数据，如情感词汇或词向量，通过聚类或降维技术来识别文本中的情感模式。主题建模方法如 latent dirichlet allocation（LDA）也可用于情感分析。

3. 深度学习方法

- ☑ 循环神经网络（RNN）：使用 RNN 或其变种，如长短时记忆网络（LSTM）或门控循环单元（GRU），来处理文本数据的时序信息，以捕获文本中的情感信息。
- ☑ 卷积神经网络（CNN）：使用 CNN 来捕获文本中的局部模式，尤其是在文本分类任务中，CNN 在情感分析中也有应用。
- ☑ 预训练模型：使用大规模的文本数据对预训练的深度学习模型（如 BERT、GPT 等）进行微调，以执行情感分析任务。

4. 混合方法

将基于规则的方法和机器学习或深度学习方法结合使用，以提高情感分析的准确性和稳定性。例如，可以使用规则引擎来处理一些明显的情感信号，然后使用机器学习模型进一步分析文本。

5. 情感词汇库

使用情感词汇库，其中包含带有情感标签的词汇，以帮助识别文本中的情感。情感得

分可以通过计算文本中情感词汇的权重和数量来生成。

6. 特征工程

在机器学习方法中，可以使用各种特征提取技术，如词袋模型（bag of words model）、TF-IDF（term frequency-inverse document frequency）和词嵌入（word embeddings）等，来表示文本数据以供模型使用。

请看下面的实例，功能是使用 TextBlob（情感分析库）分析同花顺新闻快讯的舆情是否积极、消极或中性。TextBlob 是一个 Python 库，用于自然语言处理任务，包括文本情感分析、文本分类、词性标注、命名实体识别、翻译等。TextBlob 构建在 NLTK（自然语言工具包）和 Pattern 库的基础上，提供了一个更简单和更高级的 API，使 NLP 任务更容易实现。

实例 7-1：获取同花顺新闻快讯的舆情信息（源码路径：daima/7/zhuacsv.py 和 new01.py）

（1）实例文件 zhuacsv.py 的功能是获取 TuShare 中指定日期范围内来自同花顺的新闻快讯信息，然后将获取的新闻信息保存到 CSV 文件中。具体实现代码如下所示。

```
import pandas as pd
import tushare as ts
import re  # 导入正则表达式模块

def save_news_to_csv(start_date, end_date):
    pro = ts.pro_api()
    df = pro.news(src='10jqka', start_date=start_date, end_date=end_date)

    if df.empty:
        print("没有找到新闻信息.")
        return

    # 使用正则表达式替换不允许的字符
    valid_start_date = re.sub(r'[^\w]', '_', start_date)
    valid_end_date = re.sub(r'[^\w]', '_', end_date)

    # 指定保存的文件名
    file_name = f"news_{valid_start_date}_{valid_end_date}.csv"

    # 保存为CSV文件
    df.to_csv(file_name, index=False, encoding='utf-8')
    print(f"新闻信息已保存到 {file_name}")

# 指定时间范围
start_date = '2023-01-11 09:00:00'
end_date = '2023-08-22 10:10:00'

# 调用函数保存新闻信息到CSV文件
save_news_to_csv(start_date, end_date)
```

在上述代码中，获取了指定时间范围内的同花顺新闻快讯信息，然后保存到 CSV 文件

中。因为在 Windows 系统中，文件名中不能包含以下字符之一：\ / : * ? " < > |，所以使用正则表达式对文件名进行了处理。为了便于后面的编程处理，代码执行后得到的 CSV 文件将被重新命名为 news_10jqka.csv，文件中的内容如下所示。

```
datetime,content,title
2023-08-22 10:09:00,天眼查App显示，近日，百度在线网络技术（北京）有限公司申请注册多个“灵境造极”“灵境奇点”“灵境矩阵”“灵境回声”商标，国际分类为网站服务、科学仪器，当前商标状态均为申请中。,百度申请灵境系列商标
2023-08-22 10:08:00,在全球利率上行压力的背景下，日本10年期政府债券收益率周二创下九年来新高。该收益率升至0.66%，为2014年以来的最高水平。这增加了日本央行可能进入市场进行计划外购债操作以减缓涨势的可能性。,日本10年期国债收益率创九年来新高
2023-08-22 10:07:00,在全球利率上行压力的背景下，日本10年期政府债券收益率周二创下九年来新高。该收益率升至0.66%，为2014年以来的最高水平。,日本10年期国债收益率创九年来新高
2023-08-22 10:07:00,光伏概念震荡走低，隆基股份跌超5%，精工科技、阿特斯、固德威、晶澳科技等纷纷跟跌。,光伏概念震荡走低 隆基股份跌超5%
2023-08-22 10:02:00,据工信部网站，8月21日，第29次亚太经合组织（APEC）中小企业部长会议在美国西雅图举行。会议主题是“为所有人创造有韧性和可持续的未来”。工业和信息化部副部长徐晓兰率团出席会议并围绕“通过数字化工具和技术为全球市场中的中小微企业赋能”...,工信部副部长徐晓兰率团出席第29次APEC中小企业部长会议
2023-08-22 09:59:00,传媒板块异动拉升，中国科传涨停，中原传媒、中国出版、南方传媒、浙数文化等纷纷跟涨。,传媒板块异动拉升 中国科传涨停
2023-08-22 09:58:00,企查查APP显示，近日，北京永辉超市有限公司、北京永辉商业有限公司新增一则被执行人信息，执行标的1028万余元，执行法院为北京市石景山区人民法院。企查查信息显示，北京永辉超市有限公司成立于2008年10月，注册资本6亿人民币，法定代表人为彭...,北京永辉超市被强执1028万
2023-08-22 09:57:00,据沈阳晚报，2023年铁西区（经开区、中德园）房交会于近日启动，在房交会期间，购买参展企业新建商品住房且完成登记备案的市民，都能享受契税补贴、金融、教育、优惠、“拼房团购”多重优惠政策：商品住房契税补贴50%，享有优质的教育资源及入学...,沈阳铁西区房交会：契税可享补贴50%，团购5套以上再享9.5...
2023-08-22 09:56:00,据国家统计局网站，为动态监测我国经济发展新动能变动情况，国家统计局统计科学研究所基于《新产业新业态新商业模式统计监测制度》和经济发展新动能统计指标体系，采用定基指数方法测算了2022年我国经济发展新动能指数，并修订了历史指数数据。...,国家统计局：2022年我国经济发展新动能指数比上年增长28.4%
2023-08-22 09:56:00,根据CINNO Research统计数据显示，2023年1-6月中国半导体项目投资金额约8553亿人民币，同比下滑22.7%，全球半导体产业仍处于去库存阶段。,机构：上半年中国半导体产业投资金额同比下滑22.7%
2023-08-22 09:53:00,中国建设银行发布关于调整银行承兑汇票承兑手续费收费标准的公告。自2023年9月1日起，将调整银行承兑汇票承兑手续费收费标准为按银行承兑汇票票面金额的0.05%收取。此前的原收费标准：期限3个月（含）以内，按票面金额的0.05%收取；期限3-6个月...,建行下调银行承兑汇票承兑手续费
2023-08-22 09:52:00,A股三大指数集体翻绿，两市超3000只个股下跌。,A股三大指数集体翻绿
2023-08-22 09:51:00,据央视新闻，记者从中国国家铁路集团有限公司获悉，全国铁路暑运客流持续保持高位运行，7月1日至8月21日，已累计发送旅客7.01亿人次，其中，8月19日发送旅客1568.6万人次，再次刷新暑运单日旅客发送量历史新高。全国铁路日均开行旅客列车达10444...,全国铁路暑运发送旅客已突破七亿人次
2023-08-22 09:50:00,"国防军工板块异动拉升，捷强装备涨超12%，观想科技涨超10%，恒宇信通、纵横股份、中航沈飞等跟涨。
####省略后面的内容
```

（2）编写实例文件 new01.py，功能是对文件 news_10jqka.csv 中的前 10 条数据的标题进行情感分析，并输出每个标题的情感极性（积极、消极或中性）。具体实现代码如下所示。

```
import pandas as pd
from textblob import TextBlob
import tushare as ts

# 使用tushare获取特定股票的新闻标题
def get_stock_news(stock_code, start_date, end_date):
    pro = ts.pro_api()
    # 设置数据源为同时获取sina、同花顺和东方财富的新闻
    df_sina = pro.news(src='sina', start_date=start_date, end_date=end_date, codes=
stock_code)
    df_ths = pro.news(src='ths', start_date=start_date, end_date=end_date, codes=
stock_code)
    df_eastmoney = pro.news(src='eastmoney', start_date=start_date, end_date= end_date,
codes=stock_code)

    # 合并不同数据源的新闻标题
   news_headlines = pd.concat([df_sina['title'], df_ths['title'], df_eastmoney['title']],
ignore_index=True)

    return news_headlines

# 使用TextBlob进行情感分析
def analyze_sentiment(text):
    analysis = TextBlob(text)
    if analysis.sentiment.polarity > 0:
        return "积极"
    elif analysis.sentiment.polarity < 0:
        return "消极"
    else:
        return "中性"

# 主函数
def main():
    # 指定宁德时代的股票代码和日期范围
    stock_code = "300750"  # 宁德时代的股票代码
    start_date = "2023-01-01"
    end_date = "2023-09-30"

    # 获取股票新闻标题
    news_headlines = get_stock_news(stock_code, start_date, end_date)

    # 进行情感分析并将结果添加到DataFrame中
    sentiment_results = []
    for headline in news_headlines:
        sentiment = analyze_sentiment(headline)
        sentiment_results.append(sentiment)

    # 创建DataFrame来存储新闻标题和情感分析结果
    df = pd.DataFrame({"Headline": news_headlines, "Sentiment": sentiment_results})

    # 输出情感分析结果
    print(df.head(10))  # 输出前10个新闻标题的情感分析结果
```

```
if __name__ == "__main__":
    main()
```

代码执行后会输出如下内容。

```
标题：百度申请灵境系列商标
情感：中性

标题：日本10年期国债收益率创九年来新高
情感：中性

标题：日本10年期国债收益率创九年来新高
情感：中性

标题：光伏概念震荡走低 隆基股份跌超5%
情感：中性

标题：工信部副部长徐晓兰率团出席第29次APEC中小企业部长会议
情感：中性

标题：传媒板块异动拉升　中国科传涨停
情感：中性

标题：北京永辉超市被强执1028万
情感：中性

标题：沈阳铁西区房交会：契税可享补贴50%，团购5套以上再享9.5...
情感：中性

标题：国家统计局：2022年我国经济发展新动能指数比上年增长28.4%
情感：中性

标题：机构：上半年中国半导体产业投资金额同比下滑22.7%
情感：中性
```

情感分析的结果可能受到多个因素的影响，包括文本数据的质量、模型的训练数据以及情感分析算法本身。例如，在本实例中，对新闻标题进行情感分析后得到的结果都是中性，这是因为 TextBlob 是一个基于规则和情感词典的简单情感分析工具，它可能对一些特定的文本有限制，因此在处理金融领域的文本时，可能无法准确捕捉文本的情感。通常，金融领域的情感分析需要更复杂的模型，以考虑上下文和特定领域的词汇。

为了改进情感分析的结果，可以考虑以下措施。

（1）使用更大规模的数据集。更多的数据可能有助于提高模型的性能，尤其是在特定领域或行业中。

（2）自定义情感词汇库。你可以构建或扩展情感词汇库，以更好地匹配特定行业或领域的情感表达方式。

（3）调整模型架构和参数。尝试不同的情感分析模型、算法或参数，以找到最适合宁德时代新闻标题的模型。

（4）使用金融领域的模型。金融领域的情感分析模型更具有针对性，分析结果更加准确。

情感分析在各种应用领域中都有广泛的用途，包括社交媒体监测、产品评论分析、市场研究、舆情分析、客户服务反馈分析等。选择适当的方法取决于数据的性质、任务的复杂性以及可用的计算资源。通常，使用大规模的、有标签的训练数据可以提高情感分析模型的性能，这一点将在本章后面的内容中进行讲解。

7.2 基于人工智能的金融市场情绪分析

基于人工智能的金融市场情绪分析模型使用机器学习和自然语言处理技术，旨在从大量文本数据中自动识别和分析市场参与者的情感和情绪。

7.2.1 传统情绪分析方法的局限性

传统情绪分析方法在处理文本数据中的情感时存在一些局限性，这些局限性包括以下内容。

- ☑ 单一的情感分类：传统情感分析方法通常将情感分类为积极、消极和中性三个类别，或者更少的类别。这种粗粒度的分类不能捕捉到情感的复杂性和多样性，例如，无法分辨愤怒和焦虑之间的差异。
- ☑ 上下文考虑不足：传统方法通常基于词汇的情感极性来判断情感，而没有充分考虑上下文。同样的词汇在不同上下文中可能具有不同的情感含义，这可能导致情感分析的不准确性。
- ☑ 情感表达多样性：人们在表达情感时会使用多种方式，包括隐喻、比喻、幽默和反讽等。传统情感分析方法难以处理这种多样性，因为它们通常依赖于情感词汇的匹配。
- ☑ 领域差异：情感分析模型通常需要在特定领域或语境中进行训练。这意味着在不同领域或行业中，情感分析的性能可能会下降，因为模型无法捕捉到领域特定的情感表达方式。
- ☑ 情感标注数据的稀缺性：构建情感分析模型需要大量的标注数据，特别是对于深度学习方法。然而，获取高质量的标注数据是昂贵且耗时的，因此在某些领域和语言中可能缺乏足够的训练数据。
- ☑ 情感混合：有时在文本中会包含多种情感，而传统方法难以有效处理这种情况。例如，一篇评论可能同时包含积极和消极的情感，传统方法可能只选择其中一个

情感类别。

☑ 语言变化和流行度：情感词汇和表达方式随着时间的推移和文化的变化而变化，传统情感分析方法可能无法跟上这种变化，特别是在快速演变的社交媒体环境中。

为了克服传统情感分析方法的这些局限性，市场研究人员和开发工程师开始采用深度学习方法，如循环神经网络（RNN）、卷积神经网络（CNN）和预训练的语言模型（如 BERT），这些方法可以更好地捕捉文本中的情感信息，以提高情感分析的准确性和效率。此外，研究人员还在探索多模态情感分析，将文本与图像、音频等其他模态的数据结合起来，以更全面地理解情感。

7.2.2　机器学习与情绪分析

机器学习是一种用于情绪分析的强大工具，它可以帮助自动识别、理解和分类文本或语音中包含的情感、情绪和情感倾向。机器学习在情绪分析中的主要应用和作用如下。

☑ 情感分类：机器学习模型可以用于执行情感分类任务，将文本或语音分为积极、消极、中性等情感类别。这些模型使用已标记的训练数据来学习情感模式，并在新数据上进行情感分类。

☑ 情感极性分析：机器学习可以帮助分析文本或语音中的情感极性，即情感的强度和方向。例如，确定文本中的情感是强烈的积极情感还是轻微的消极情感。

☑ 特征工程：在情感分析中，机器学习可以用于提取文本或语音数据的相关特征，如词频、TF-IDF、词嵌入等。这些特征可以用于训练情感分类模型。

☑ 模型选择：机器学习提供了多种可用于情感分析的模型，包括朴素贝叶斯、支持向量机、决策树、随机森林、深度神经网络等。选择合适的模型取决于任务的复杂性和数据的性质。

☑ 训练和调优：机器学习模型需要在带有情感标签的训练数据上进行训练，然后使用交叉验证等技术进行性能评估和超参数调优，以提高模型的准确性和泛化能力。

☑ 多语言支持：机器学习方法可以用于多语言情感分析，使其适用于不同语言的文本和语音数据。

☑ 实时情感分析：机器学习模型可以部署为实时情感分析系统，可以实时监测社交媒体、新闻流、客户服务反馈等数据源中的情感信息。

☑ 自动情感标签生成：机器学习方法还可以用于生成情感标签，如使用情感词汇库和上下文信息来标记文本中的情感。

注意：虽然机器学习在情感分析中表现出色，但它仍然面临一些挑战。例如，需要大量的标记数据来训练模型，标签数据的质量可能会影响模型性能。此外，机器学习模型通常需要大量计算资源和时间来进行训练和调优。近年来，预训练的语言模型（例如BERT、GPT）已经在情感分析中取得了显著的突破，提高了情感分析的准确性和效率。

7.3 预训练模型：BERT

预训练模型（pretrained models）是通过在大规模文本数据上进行自监督学习训练的深度学习模型。这些模型通过学习通用的语言表示，可以在各种 NLP 任务中提供出色的性能。在市场中有很多和金融情绪预测相关的预训练模型，如 BERT 和 FinBERT 等。

7.3.1 BERT 模型介绍

BERT 是一种预训练的 Transformer 模型，它通过在大规模文本数据上进行自监督学习来学习通用的语言表示。BERT 可以理解文本中的上下文信息，并生成文本的上下文相关表示。在使用 BERT 进行情感分析任务时，通常会将其用作特征提取器，然后将提取的特征输入分类器中进行情感分类。BERT 需要进一步微调以适应特定的情感分析任务。使用 BERT 的基本流程如下。

（1）选择深度学习框架。需要选择一个深度学习框架，如 TensorFlow 或 PyTorch，将其作为实际加载和使用 BERT 模型的工具。这些框架提供了用于加载和操作预训练模型的 API。

（2）下载预训练模型权重。BERT 的预训练权重通常可以从 Hugging Face Transformers 库（https://huggingface.co/models）或官方发布的源获取。你可以选择下载适合你任务的特定版本的 BERT 模型权重。

（3）加载模型权重。使用所选的深度学习框架，加载下载的 BERT 模型权重。在 TensorFlow 中，你可以使用 TensorFlow Hub 或 TensorFlow 的 Keras 模型加载 BERT。在 PyTorch 中，你可以使用 Hugging Face Transformers 库加载 BERT。

（4）使用模型。一旦加载了 BERT 模型权重，就可以使用它来进行文本编码、特征提取或执行各种 NLP 任务，如情感分析、文本分类等。

请看下面的实例，功能是使用预训练的 BERT 模型对同花顺中的新闻快讯标题进行情感分析。在使用预训练的 BERT 模型之前，需要先登录其官方网站下载，本实例使用的是 chinese_wwm_ext_pytorch 简约版。下载成功后将其保存到 path 目录。

实例 7-2：使用 BERT 获取同花顺新闻快讯的舆情信息（源码路径：daima/7/new02.py）

实例文件 new02.py 的具体实现代码如下所示。

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification
import pandas as pd

# 定义模型和tokenizer的文件路径
model_path = "path"  # 替换为你的模型路径
tokenizer_path = "path"  # 替换为你的tokenizer路径

# 加载BERT tokenizer和模型
```

```
tokenizer = BertTokenizer.from_pretrained(tokenizer_path)
model = BertForSequenceClassification.from_pretrained(model_path)

# 读取CSV文件
df = pd.read_csv("news_10jqka.csv")

# 取前10条数据
df = df.head(10)

# 遍历每条数据的title列进行情感分析
for index, row in df.iterrows():
    text = row["title"]

    # 使用tokenizer将文本编码成模型可接受的格式
    input_ids = tokenizer.encode(text, add_special_tokens=True, max_length=128,
padding="max_length", truncation=True,
                                 return_tensors="pt")

    # 使用模型进行情感分析
    with torch.no_grad():
        outputs = model(input_ids)
        logits = outputs.logits

    # 获取情感分析的结果（这里假设你的模型是二分类，积极/消极）
    predicted_class = torch.argmax(logits, dim=1).item()
    sentiment = "积极" if predicted_class == 1 else "消极" if predicted_class == 0
else "中性"

    # 打印情感分析结果
    print(f"标题：{text}")
    print(f"情感：{sentiment}")
```

在上述代码中，首先读取 CSV 文件的内容，然后遍历前 10 条数据的标题。然后使用 BERT 模型进行情感分析，并将结果存储在 sentiment 列表中。最后，将情感分析结果添加到原始 DataFrame 中，并打印出来。代码执行后会输出如下内容。

```
标题：百度申请灵境系列商标
情感：积极
标题：日本10年期国债收益率创九年来新高
情感：积极
标题：日本10年期国债收益率创九年来新高
情感：积极
标题：光伏概念震荡走低 隆基股份跌超5%
情感：积极
标题：工信部副部长徐晓兰率团出席第29次APEC中小企业部长会议
情感：积极
标题：传媒板块异动拉升　中国科传涨停
情感：积极
标题：北京永辉超市被强执1028万
情感：积极
标题：沈阳铁西区房交会：契税可享补贴50%，团购5套以上再享9.5...
```

```
情感：积极
标题：国家统计局：2022年我国经济发展新动能指数比上年增长28.4%
情感：积极
标题：机构：上半年中国半导体产业投资金额同比下滑22.7%
情感：积极
```

7.3.2 情感关键字

在前面的实例 7-2 的输出结果中，所有的标题都被分类为"积极"情感，这可能是因为使用的预训练模型在情感分析任务上并没有很好地适应数据，或者数据集中的标题文本在情感上确实偏向积极。接下来需要改进实例，例如，根据标题中包含的特定关键字来进行情感分类，将包含 negative_keywords = ["利空"，"下跌"，"跌幅"，"亏损"，"超跌"，"下滑"，"强执"，"跌超"]的标题分类为消极情感，将包含 positive_keywords = ["利好"，"增长"，"涨幅"，"上涨"，"涨停"]]的标题分类为积极情感，下面的实例演示了这一功能。

实例 7-3：添加情感关键字（源码路径：daima/7/new03.py）

实例文件 new03.py 的具体实现代码如下所示。

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification
import pandas as pd

# 加载预训练的BERT模型和tokenizer
model_path = "path"  # 替换为你的预训练模型路径
tokenizer_path = "path"  # 替换为你的预训练tokenizer路径
tokenizer = BertTokenizer.from_pretrained(tokenizer_path)
model = BertForSequenceClassification.from_pretrained(model_path)

# 读取新闻数据
df = pd.read_csv("news_10jqka.csv")
titles = df["title"][:10]  # 选择前10条标题进行情感分析

# 定义情感关键字
positive_keywords = ["利好", "增长", "涨幅", "上涨", "涨停"]
negative_keywords = ["利空", "下跌", "跌幅", "亏损", "超跌", "下滑", "强执", "跌超"]

# 遍历标题并进行情感分析
for title in titles:
    # 使用tokenizer将文本编码成模型可接受的格式
    input_ids = tokenizer.encode(title, add_special_tokens=True, max_length=128,
padding="max_length", truncation=True,
                                 return_tensors="pt")

    # 检查标题是否包含消极关键字，如果包含则将情感类别设置为消极
    contains_negative_keyword = any(keyword in title for keyword in
negative_keywords)

    # 检查标题是否包含积极关键字，如果包含则将情感类别设置为积极
```

```
    contains_positive_keyword = any(keyword in title for keyword in
positive_keywords)

    # 使用模型进行情感分析
    with torch.no_grad():
        outputs = model(input_ids)
        logits = outputs.logits

    # 获取情感分析的结果
    predicted_class = torch.argmax(logits, dim=1).item()

    if contains_negative_keyword:
        sentiment = "消极"
    elif contains_positive_keyword:
        sentiment = "积极"
    else:
        if predicted_class == 1:
            sentiment = "积极"
        elif predicted_class == 0:
            sentiment = "消极"
        else:
            sentiment = "中性"

    # 打印情感分析结果
    print(f"标题：{title}")
    print(f"情感：{sentiment}")
    print()
```

上述代码的实现流程如下。

（1）导入必要的库，包括 PyTorch、Hugging Face Transformers 库和 Pandas。

（2）定义预训练模型和 tokenizer 的文件路径，你需要将其替换为实际的路径。这些路径指向了已经下载的 BERT 模型和 tokenizer。

（3）加载 BERT tokenizer 和模型。从预训练文件夹中加载预训练的 BERT 模型和 tokenizer。

（4）定义情感关键字，包括 negative_keywords 和 positive_keywords，用于后处理。

（5）读取包含新闻标题的 CSV 文件（假设文件名为 news_10jqka.csv），并将其存储在 DataFrame 中。

（6）对前 10 条数据进行情感分析，遍历 DataFrame 的前 10 行数据。对每个新闻标题进行情感分析的主要步骤如下。

☑ 使用 tokenizer 将标题文本编码为模型可接受的格式。

☑ 使用模型进行情感分析，并获取模型的输出。

☑ 根据输出的结果，判断标题的情感是积极、消极还是中性。

☑ 通过检查标题中是否包含情感关键字，进行后处理以更改情感标签。

（7）打印输出每个标题的情感分析结果，包括标题文本和情感标签。

执行后会输出如下内容。

```
标题：百度申请灵境系列商标
情感：积极

标题：日本10年期国债收益率创九年来新高
情感：积极

标题：日本10年期国债收益率创九年来新高
情感：积极

标题：光伏概念震荡走低 隆基股份跌超5%
情感：消极

标题：工信部副部长徐晓兰率团出席第29次APEC中小企业部长会议
情感：积极

标题：传媒板块异动拉升　中国科传涨停
情感：积极

标题：北京永辉超市被强执1028万
情感：消极

标题：沈阳铁西区房交会：契税可享补贴50%，团购5套以上再享9.5...
情感：积极

标题：国家统计局：2022年我国经济发展新动能指数比上年增长28.4%
情感：积极

标题：机构：上半年中国半导体产业投资金额同比下滑22.7%
情感：消极
```

虽然上面代码的识别结果符合我们的预期，但是在上面的代码中添加的情感关键字 negative_keywords 和 positive_keywords 与大模型完全无关，也不是预训练模型的参数，它们只是在代码中定义的两个列表，用于指定哪些关键字应该被视为负面情感和正面情感的指示标志。

预训练模型（如 BERT）本身并不包含关于具体情感词汇的信息，它是根据大规模文本数据进行预训练的，可以理解为它具备了一定的语言理解能力，但不具备特定情感的识别能力。因此，为了对特定情感进行分类，我们需要自行定义这些情感关键字，并根据它们在文本中的出现来进行分类。

在上述代码中，negative_keywords 和 positive_keywords 用于识别标题中是否包含特定情感相关的关键字，以决定将标题分类为消极情感还是积极情感。这些关键字是你自己定义的，不是预训练模型的一部分。

7.3.3 模型微调

模型微调（fine-tuning）是深度学习中一种常见的训练策略，它通常指的是在预训练模

型的基础上，通过在特定任务的数据集上进行额外的训练来适应特定任务。微调的核心思想是利用已经在大规模数据上预训练好的模型的知识，然后通过在任务相关的数据上进行有监督的训练，进一步优化模型的性能。

例如，在前面的实例中，情感关键字 negative_keywords 和 positive_keywords 通常不会直接添加到预训练模型中，因为预训练模型的参数是通过大规模文本数据进行学习的，而不是通过手动添加特定的关键字。这些关键字可以在微调阶段或应用阶段的代码中使用，以帮助模型对特定情感或主题进行分类。

在情感分析任务中，通常的做法是将情感关键字用于后续的数据预处理或后处理步骤，具体如下。

- ☑ 数据预处理：在训练数据中，可以根据这些关键字标记文本样本的标签，使模型在训练过程中了解这些情感关键字与标签之间的关系。
- ☑ 后处理：在模型输出的预测结果中，你可以根据这些关键字对预测结果进行调整，以提高情感分析的准确性。例如，如果输出结果包含“利好”或“增长”等关键字，你可以将其分类为积极情感。

实例 7-4：为预训练模型添加情感关键字（源码路径：daima/7/wei.py）

实例文件 wei.py 的具体实现代码如下所示。

```
import torch
import torch.nn as nn
from transformers import BertTokenizer, BertForSequenceClassification
from torch.utils.data import DataLoader, TensorDataset
import pandas as pd

# 加载预训练的BERT模型和tokenizer
model_path = "path"  # 替换为你的预训练模型路径
tokenizer_path = "path"  # 替换为你的tokenizer路径
tokenizer = BertTokenizer.from_pretrained(tokenizer_path)
model = BertForSequenceClassification.from_pretrained(model_path, num_labels=3)  #
假设有3个情感类别

# 读取带有情感标签的CSV文件
df = pd.read csv("news with sentiment.csv")
texts = df["title"]
label_mapping = {"消极": 0, "积极": 1, "中性": 2}  # 情感标签到数值的映射
label_values = df["qinggan"].map(label_mapping)  # 将字符串标签映射为数值标签

# 数据预处理
tokenized_texts   =   tokenizer(texts.tolist(),   padding=True,   truncation=True,
return_tensors="pt")
input_ids = tokenized_texts["input_ids"]
attention_mask = tokenized_texts["attention_mask"]

# 创建数据加载器
dataset = TensorDataset(input_ids, attention_mask, torch.tensor(label_values))  #
修改此处为torch.tensor(label_values)
```

```
batch_size = 2
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)

# 微调模型
num_epochs = 3
for epoch in range(num_epochs):
    for batch in dataloader:
        input_ids, attention_mask, labels = batch
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

# 保存微调后的模型
model.save_pretrained("fine_tuned_model")
```

上述代码的实现流程如下。

（1）加载预训练 BERT 模型和 tokenizer。

model_path 和 tokenizer_path 分别是预训练模型和 tokenizer 的路径。BertTokenizer. from_pretrained（tokenizer_path）用于加载 BERT 模型的 tokenizer。BertForSequence Classification. from_pretrained（model_path，num_labels=3）用于加载 BERT 模型，num_labels 设置为 3，表示假设有 3 个情感类别。

（2）读取带有情感标签的 CSV 文件。

使用 pd.read_csv（"news_with_sentiment.csv"）读取包含标题和情感标签的 CSV 文件。变量 texts 包含 CSV 文件中的标题数据。label_mapping 是从情感标签到数值的映射，例如，“消极”映射为 0，“积极”映射为 1，“中性”映射为 2。label_values 通过将字符串标签映射为数值标签创建了一个新的标签列。

（3）数据预处理。

使用 tokenizer 将文本数据编码为模型可接受的格式，包括标记化（tokenization）、填充（padding）和截断（truncation）等操作。创建了 input_ids 和 attention_mask 以准备输入到模型中。

（4）创建数据加载器。

TensorDataset 用于将输入数据、注意力掩码和标签封装成 PyTorch 数据集。batch_size 指定了每个训练批次的大小。DataLoader 用于生成批次数据以供模型训练。

（5）定义损失函数和优化器。

使用 nn.CrossEntropyLoss()定义了交叉熵损失函数，用于多类别分类任务。使用 torch.optim.AdamW 定义了 AdamW 优化器。

（6）微调模型。

使用嵌套的循环进行微调，外层循环控制训练的轮数，内层循环用于逐批次处理训练数据。在每个批次中，将输入数据和标签传递给模型，并计算损失。然后通过反向传播（backward()）和优化器来更新模型的权重。这个过程会在多个轮次中重复，以提高模型性能。

（7）保存微调后的模型：使用 model.save_pretrained（"fine_tuned_model"）保存微调后的模型，可以后续用于推断任务。

7.4　预训练模型：FinBERT

FinBERT 是基于 BERT 的模型，但经过了针对金融领域的预训练和微调。使用了金融领域的大量文本数据进行预训练，从而使其在金融文本分类和情感分析等任务中表现出色。

7.4.1　FinBERT 模型介绍

FinBERT 模型的官方地址是 https://github.com/ProsusAI/finBERT，大家可以使用如下 git 命令将 FinBERT 下载到你的本地计算机，或者直接从 GitHub 页面下载 ZIP 文件。

```
git clone https://github.com/yiyanghkust/FinBERT.git
```

在网络结构上，FinBERT 采用了与 Google 发布的原生 BERT 相同的架构，包含了 FinBERT-Base 和 FinBERT-Large 两个版本，其中前者采用了 12 层 Transformer 结构，后者采用了 24 层 Transformer 结构。FinBERT 采用两大类预训练任务，分别是字词级别的预训练和任务级别的预训练。其中，在任务级别的预训练上，为了让模型更好地学习语义层的金融领域知识，更全面地学习金融领域词句的特征分布，我们同时引入了两类有监督学习任务，分别是研报行业分类和财经新闻的金融实体识别任务，具体说明如下。

（1）研报行业分类。

对于公司点评、行业点评类的研报，天然具有很好的行业属性，因此我们利用这类研报自动生成了大量带有行业标签的语料。并据此构建了行业分类的文档级有监督任务，各行业类别语料在 5k~20k 之间，共计约 40 万条文档级语料。

（2）财经新闻的金融实体识别。

与研报行业分类任务类似，利用已有的企业工商信息库以及公开可查的上市公司董监高信息，基于金融财经新闻构建了命名实体识别类的任务语料，共包含有 50 万条的有监督语料。

7.4.2　基于 FinBERT 模型的市场情感分析系统

请看下面的实例，演示了使用预训练的深度学习模型 FinBERT 进行金融情感分类任务的过程，包括数据处理、模型微调、性能评估和预测。这有助于金融领域中对新闻标题等

文本数据进行情感分析和情感预测。

实例 7-5：金融市场情感分析（源码路径：daima/7/financial-classification.ipynb）

概括来说，本实例实现了以下功能。

- ☑ 加载预训练的金融情感分类模型（FinBERT）。
- ☑ 对金融新闻标题进行数据预处理，包括分词、编码、数据拆分等。
- ☑ 使用训练数据对模型进行微调（fine-tuning）以适应金融情感分类任务。
- ☑ 通过训练和验证过程，监测模型性能并选择最佳模型。
- ☑ 加载最佳模型，并使用其进行情感分类预测。
- ☑ 计算并打印模型在验证集上的准确率和每个情感类别的准确率。
- ☑ 提供了一种在金融领域进行情感分类任务的示例，可以用于预测金融新闻标题的情感倾向。

本实例的具体实现流程如下所示。

（1）函数 show_headline_distribution（sequence_lengths，figsize=（15，8））用于分析新闻标题长度的分布情况，通过可视化直方图展示新闻标题长度的分布，以便了解标题长度的数据分布特征。它还计算了长度大于 512 的标题所占的百分比，提供了对极长标题的信息。具体实现代码如下所示。

```
# 显示新闻标题长度的分布
def show_headline_distribution(sequence_lengths, figsize=(15, 8)):
    # 获取长度大于512的新闻标题所占的百分比
    len_512_plus = [rev_len for rev_len in sequence_lengths if rev_len > 512]
    percent = (len(len_512_plus) / len(sequence_lengths)) * 100

    print("最大序列长度为 {}".format(max(sequence_lengths)))

    # 配置图表大小
    plt.figure(figsize=figsize)

    sns.set(style='darkgrid')

    # 增加图表上的信息
    sns.set(font_scale=1.3)

    # 绘制结果
    sns.distplot(sequence_lengths, kde=False, rug=False)
    plt.title('新闻标题长度分布')

    plt.xlabel('新闻标题长度')
    plt.ylabel('新闻标题数量')
```

（2）函数 show_random_headlines（total_number，df）用于随机选择指定数量的新闻标题，并打印它们的情感标签和标题文本。这有助于查看数据集中的随机样本，以了解情感标签和标题之间的关系。具体实现代码如下所示。

```
# 显示随机新闻标题
def show_random_headlines(total_number, df):
    # 随机抽取一定数量的新闻标题
    n_reviews = df.sample(total_number)

    # 打印每个新闻标题
    for val in list(n_reviews.index):
        print("新闻 #{}".format(val))
        print(" - 情感: {}".format(df.iloc[val]["sentiment"]))
        print(" - 新闻标题: {}".format(df.iloc[val]["NewsHeadline"]))
        print("")
```

（3）函数 get_headlines_len（df）的主要功能是计算每个新闻标题的长度，并返回一个包含所有标题长度的列表。它通过对新闻标题进行编码来实现，以便后续的情感分析任务可以使用这些编码后的数据。具体实现代码如下所示。

```
# 获取新闻标题的长度
def get_headlines_len(df):
    headlines_sequence_lengths = []
    print("编码中...")
    for headline in tqdm(df.NewsHeadline):
        encoded_headline = finbert_tokenizer.encode(headline, add_special_tokens=
True)

        # 记录编码后的新闻标题长度
        headlines_sequence_lengths.append(len(encoded_headline))
    print("任务结束.")

    return headlines_sequence_lengths
```

（4）函数 encode_sentiments_values（df）用于将情感标签（如“positive”、“negative”）编码为数字，以便在深度学习模型中进行处理。它创建一个情感标签到数字编码的映射，并将数据集中的情感标签替换为相应的数字。具体实现代码如下所示。

```
# 编码情感值
def encode_sentiments_values(df):
    possible_sentiments = df.sentiment.unique()
    sentiment_dict = {}

    for index, possible_sentiment in enumerate(possible_sentiments):
        sentiment_dict[possible_sentiment] = index

    # 编码所有情感值
    df['label'] = df.sentiment.replace(sentiment_dict)

    return df, sentiment_dict
```

（5）函数 f1_score_func（preds，labels）的功能是计算 F1 得分，用于评估模型在情感分类任务中的性能。它通过比较模型的预测结果和真实标签来计算 F1 得分，考虑了精确度

和召回率。具体实现代码如下所示。

```
# F1得分函数
def f1_score_func(preds, labels):
    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return f1_score(labels_flat, preds_flat, average='weighted')
```

（6）函数 accuracy_per_class（preds，labels）：用于计算每个情感类别的分类准确度。它将模型的预测结果与真实标签进行比较，并为每个情感类别计算准确度，以评估模型在不同情感类别上的性能。具体实现代码如下所示。

```
# 每个类别的准确度
def accuracy_per_class(preds, labels):
    label_dict_inverse = {v: k for k, v in sentiment_dict.items()}

    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()

    for label in np.unique(labels_flat):
        y_preds = preds_flat[labels_flat==label]
        y_true = labels_flat[labels_flat==label]
        print(f'类别: {label_dict_inverse[label]}')
        print(f'准确度: {len(y_preds[y_preds==label])}/{len(y_true)}\n')
```

（7）函数 evaluate（dataloader_val）用于评估模型的性能，包括计算验证集上的损失和生成预测结果。它在模型训练过程中用于监测模型的性能，并返回损失值、预测结果和真实标签，以便进一步的分析和评估。具体实现代码如下所示。

```
# 评估函数
def evaluate(dataloader_val):
    model.eval()

    loss_val_total = 0
    predictions, true_vals = [], []

    for batch in dataloader_val:
        batch = tuple(b.to(device) for b in batch)

        inputs = {'input_ids':      batch[0],
                  'attention_mask': batch[1],
                  'labels':         batch[2],
                 }

        with torch.no_grad():
            outputs = model(**inputs)

        loss = outputs[0]
        logits = outputs[1]
        loss_val_total += loss.item()
```

```
        logits = logits.detach().cpu().numpy()
        label_ids = inputs['labels'].cpu().numpy()
        predictions.append(logits)
        true_vals.append(label_ids)

    loss_val_avg = loss_val_total/len(dataloader_val)

    predictions = np.concatenate(predictions, axis=0)
    true_vals = np.concatenate(true_vals, axis=0)
    return loss_val_avg, predictions, true_vals
```

（8）使用库 Pandas 从指定路径加载 CSV 文件，并将其存储在名为 financial_data 的 DataFrame 中。具体实现代码如下所示。

```
path_to_file = "../input/financialnewsheadline/FinancialNewsHeadline.csv"
financial_data = pd.read_csv(path_to_file, encoding='latin-1', names=['sentiment',
'NewsHeadline'])

financial_data.head()
```

在文件 FinancialNewsHeadline.csv 中包含了多个新闻标题及其情感标签，每个新闻标题都附带一个情感标签，标记了新闻标题的情感极性，包括 neutral（中性）、negative（负面）和 positive（正面）。这些数据可以用于情感分析或其他自然语言处理任务的训练和分析。代码执行后会输出如下内容。

```
          sentiment   NewsHeadline
0   neutral  According to Gran , the company has no plans t...
1   neutral  Technopolis plans to develop in stages an area...
2   negative The international electronic industry company ...
3   positive With the new production plant the company woul...
4   positive According to the company 's updated strategy f...
```

（9）打印输出关数据集的一些基本信息，包括数据的形状和情感标签的分布。具体实现代码如下所示。

```
print("Data shape: {}".format(financial_data.shape))
print("\nSentiment distribution: {}".format(financial_data.sentiment.value_counts
()))
```

对上述代码的具体说明如下。

- ☑ print("Data shape: {}".format(financial_data.shape))：打印数据集的形状，即行数和列数。这将显示数据集中有多少行和多少列。
- ☑ print("\nSentiment distribution: {}".format(financial_data.sentiment.value_counts()))：打印情感标签的分布情况。value_counts()函数用于计算每个不同情感标签的出现次数，以便了解数据集中每个情感标签的样本数量。在打印之前，通过\n 添加一个换行符，以使输出内容更易读。

执行后会输出如下内容。

```
Data shape: (4846, 2)

Sentiment distribution: neutral    2879
positive   1363
negative    604
Name: sentiment, dtype: int64
```

（10）绘制情感标签的分布情况图表，以可视化展示数据集中每个情感标签的样本数量。具体实现代码如下所示。

```
plt.figure(figsize = (15,8))

sns.set(style='darkgrid')

sns.set(font_scale=1.3)
sns.countplot(x='sentiment', data = financial_data)
plt.title('News Sentiment Distribution')
plt.xlabel('News Polarity')
plt.ylabel('Number of News')
```

代码执行后会生成一个柱状图，如图 7-1 所示。这样就直观地展示了每种情感标签在数据集中的分布情况，有助于研究人员了解数据集中不同情感标签的相对频率。这对于数据的初步探索性分析（EDA）非常有用。

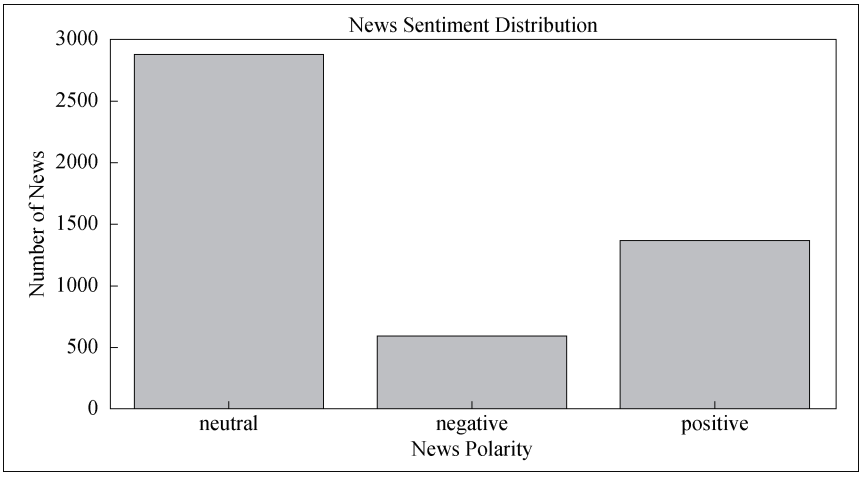

图 7-1　每种情感标签在数据集中的分布情况

（11）调用之前定义的 show_random_headlines 函数，以显示随机选择的 5 个新闻标题以及它们的情感标签。具体实现代码如下所示。

```
show_random_headlines(5, financial_data)
```

代码执行后将随机选择 5 个样本，并打印每个样本的情感标签和新闻标题文本，以便我们查看这些数据的样本内容和情感标签。

（12）调用前面定义的 encode_sentiments_values 函数，将其返回的结果存储在名为 financial_data 的 DataFrame 中，并获得了情感标签到数字编码的映射 sentiment_dict。具体实现代码如下所示。

```
financial_data, sentiment_dict = encode_sentiments_values(financial_data)
financial_data.head()
```

代码执行后会显示经编码后的数据集的前几行，以便我们查看情感标签已经以数字形式表示的数据。这有助于后续的建模和分析工作，因为模型通常需要处理数字形式的标签而不是文本标签。

（13）使用函数 train_test_split 从数据集中划分训练集（X_train 和 y_train）和验证集（X_val 和 y_val）。这些数据集可以用于模型的训练和验证，以评估模型的性能。具体实现代码如下所示。

```
X_train, X_val, y_train, y_val = train_test_split(financial_data.index.values,
                                                  financial_data.label.values,
                                                  test_size = 0.15,
                                                  random_state = 2022,
                                                  stratify = financial_data.label.values)
```

（14）在 financial_data DataFrame 中为训练集（X_train）和验证集（X_val）的样本添加一个名为 data_type 的新列，并设置其值为 train 和 val，以表示每个样本属于训练集或验证集。然后，计算每种情感标签在不同数据类型（训练集或验证集）中的出现次数。具体实现代码如下所示。

```
financial_data.loc[X_train, 'data_type'] = 'train'
financial_data.loc[X_val, 'data_type'] = 'val'
financial_data.groupby(['sentiment', 'label', 'data_type']).count()
```

代码执行后会输出如下内容。

```
             NewsHeadline
sentiment    label    data_type
negative 1   train    513
val  91
neutral  0   train    2447
val  432
positive 2   train    1159
              val     204
```

通过执行上述代码，可以查看不同情感标签在训练集和验证集中的分布情况，以确保数据在不同数据类型之间的分布是合理的。这有助于了解数据集的分层情况，以便进行合理的模型训练和验证。

（15）开始查看新闻标题长度的分布，以确定在进行数据编码时需要设定的最大长度。首先使用库 Hugging Face Transformers 中的 BertTokenizer 类，从预训练模型 ProsusAI/finbert 中加载一个金融领域的文本分词器（Tokenizer）。具体实现代码如下所示。

```
finbert_tokenizer = BertTokenizer.from_pretrained("ProsusAI/finbert", do_lower_case=True)
```

代码执行后会输出如下内容。

```
Downloading: 100%|██████████|226k/226k [00:00<00:00, 807kB/s]
Downloading: 100%|██████████|112/112 [00:00<00:00, 4.33kB/s]
Downloading: 100%|██████████|252/252 [00:00<00:00, 10.1kB/s]
Downloading: 100%|██████████|758/758 [00:00<00:00, 11.3kB/s]
```

（16）调用函数 get_headlines_len 计算数据集中每个新闻标题的长度，并将结果存储在名为 headlines_sequence_lengths 的列表中。具体实现代码如下所示。

```
headlines_sequence_lengths = get_headlines_len(financial_data)
```

代码执行后会输出如下内容。

```
Encoding in progress...
100%|██████████| 4846/4846 [00:04<00:00, 1136.56it/s]
End of Task.
```

这个列表可以用于分析新闻标题长度的分布情况，也可以用于确定在进行文本编码时需要设置的最大序列长度，以便在模型训练中进行适当的填充或截断操作。

（17）调用之前定义的 show_headline_distribution 函数，用于显示新闻标题长度的分布情况。具体实现代码如下所示。

```
show_headline_distribution(headlines_sequence_lengths)
```

代码执行后的效果如图 7-2 所示。这个直方图有助于可视化新闻标题长度的分布，从而让我们更好地了解新闻标题的长度范围和分布情况。

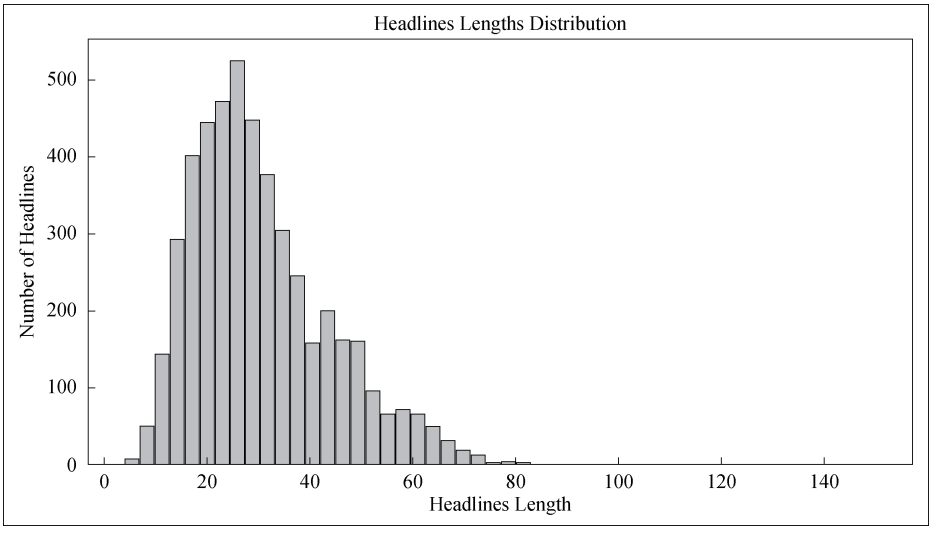

图 7-2　新闻标题长度的分布情况

（18）对训练集和验证集的新闻标题数据进行编码和处理，以准备用于金融情感分类模型的训练和验证，以便进行后续的模型训练和评估。具体实现代码如下所示。

```
encoded_data_train = finbert_tokenizer.batch_encode_plus(
    financial_data[financial_data.data_type=='train'].NewsHeadline.values,
    return_tensors='pt',
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=150 # the maximum lenght observed in the headlines
)

encoded_data_val = finbert_tokenizer.batch_encode_plus(
    financial_data[financial_data.data_type=='val'].NewsHeadline.values,
    return_tensors='pt',
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=150 # the maximum lenght observed in the headlines
)


input_ids_train = encoded_data_train['input_ids']
attention_masks_train = encoded_data_train['attention_mask']
labels_train = torch.tensor(financial_data[financial_data.data_type=='train'].label.
values)

input_ids_val = encoded_data_val['input_ids']
attention_masks_val = encoded_data_val['attention_mask']
sentiments_val = torch.tensor(financial_data[financial_data.data_type=='val'].label.
values)


dataset_train = TensorDataset(input_ids_train, attention_masks_train, labels_train)
dataset_val = TensorDataset(input_ids_val, attention_masks_val, sentiments_val)

len(sentiment_dict)
```

代码执行后会输出如下内容。

```
Truncation was not explicitly activated but `max_length` is provided a specific
value, please use `truncation=True` to explicitly truncate examples to max length.
Defaulting to 'longest_first' truncation strategy. If you encode pairs of sequences
(GLUE-style) with the tokenizer you can select this strategy more precisely by
providing a specific strategy to `truncation`.

3
```

（19）使用库 Hugging Face Transformers 中的类 AutoModelForSequenceClassification，从预训练模型 ProsusAI/finbert 中加载一个用于序列分类（Sequence Classification）任务的模型。具体实现代码如下所示。

```
model = AutoModelForSequenceClassification.from_pretrained("ProsusAI/finbert",
                                                           num_labels=len(sentiment_dict),
                                                           output_attentions=False,
                                                           output_hidden_states=False)
```

代码执行后会输出如下内容。

```
Downloading: 100%|██████████418M/418M [00:19<00:00, 23.5MB/s]
```

通过加载这个模型，可以将其应用于金融情感分类任务，模型已经包含了预训练的权重和结构，只需要进行微调（fine-tuning）以适应特定的任务即可。模型的输出将包括情感分类的预测结果。

（20）创建训练集和验证集的数据加载器（DataLoader），以便在模型训练和验证过程中批量加载数据，并在模型训练和验证过程中加载数据批次以进行前向传播和反向传播。具体实现代码如下所示。

```
batch_size = 5
dataloader_train = DataLoader(dataset_train,
                          sampler=RandomSampler(dataset_train),
                          batch_size=batch_size)

dataloader_validation = DataLoader(dataset_val,
                               sampler=SequentialSampler(dataset_val),
                               batch_size=batch_size)
```

（21）设置优化器和学习率调度器，以及指定训练的总周期数（epochs）。通过这些设置，可以在训练模型时使用 AdamW 优化器，并在每个训练周期结束后，根据学习率调度器动态调整学习率。具体实现代码如下所示。

```
optimizer = AdamW(model.parameters(),
                  lr=1e-5,
                  eps=1e-8)
epochs = 2
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,

num_training_steps=len(dataloader_train)*epochs)
```

（22）开始训练金融情感分类模型，整个训练过程包括多个训练周期。在每个训练周期内，模型在训练数据上进行前向传播、损失计算和反向传播，然后更新模型参数。同时，会计算并打印训练损失、验证损失和 F1 分数，以监测模型的性能。模型的参数在每个训练周期结束后被保存，以便后续的使用。具体实现代码如下所示。

```
seed_val = 2022
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

for epoch in tqdm(range(1, epochs+1)):
    model.train()
    loss_train_total = 0
    progress_bar = tqdm(dataloader_train, desc='Epoch {:1d}'.format(epoch),
leave=False, disable=False)
    for batch in progress_bar:
        model.zero_grad()
        batch = tuple(b.to(device) for b in batch)
        inputs = {'input_ids':     batch[0],
                  'attention_mask': batch[1],
                  'labels':       batch[2],
                 }

        outputs = model(**inputs)
        loss = outputs[0]
        loss_train_total += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()
        progress_bar.set_postfix({'training_loss':
'{:.3f}'.format(loss.item()/len(batch))})
    torch.save(model.state_dict(), f'finetuned_finBERT_epoch_{epoch}.model')
    tqdm.write(f'\nEpoch {epoch}')

    loss_train_avg = loss_train_total/len(dataloader_train)
    tqdm.write(f'Training loss: {loss_train_avg}')
    val_loss, predictions, true_vals = evaluate(dataloader_validation)
    val_f1 = f1_score_func(predictions, true_vals)
    tqdm.write(f'Validation loss: {val_loss}')
    tqdm.write(f'F1 Score (Weighted): {val_f1}')
```

代码执行后会输出显示训练过程。

```
  0%|          | 0/2 [00:00<?, ?it/s]
Epoch 1:   0%|          | 0/824 [00:00<?, ?it/s]
Epoch 1:   0%|          | 0/824 [00:01<?, ?it/s, training_loss=1.028]
Epoch 1:   0%|          | 1/824 [00:01<13:58,  1.02s/it, training_loss=1.028]
Epoch 1:   0%|          | 1/824 [00:01<13:58,  1.02s/it, training_loss=0.840]
Epoch 1:   0%|          | 2/824 [00:01<06:44,  2.03it/s, training_loss=0.840]
Epoch 1:   0%|          | 2/824 [00:01<06:44,  2.03it/s, training_loss=0.459]
Epoch 1:   0%|          | 3/824 [00:01<04:24,  3.10it/s, training_loss=0.459]
Epoch 1:   0%|          | 3/824 [00:01<04:24,  3.10it/s, training_loss=0.382]
Epoch 1:   0%|          | 4/824 [00:01<03:18,  4.13it/s, training_loss=0.382]
Epoch 1:   0%|          | 4/824 [00:01<03:18,  4.13it/s, training_loss=0.300]
#######省略部分过程
Epoch 1: 100%| ■■■■■■■■■■ | 824/824 [01:39<00:00,    8.74it/s,
training_loss=0.006]
  0%|          | 0/2 [01:40<?, ?it/s]
```

```
Epoch 1
Training loss: 0.4581567718019004
######省略部分过程
 50%|██████     | 1/2 [01:44<01:44, 104.72s/it]
Validation loss: 0.3778555450533606
F1 Score (Weighted): 0.8753212258177056
Epoch 2:   0%|          | 0/824 [00:00<?, ?it/s]
Epoch 2:   0%|          | 0/824 [00:00<?, ?it/s, training_loss=0.004]
Epoch 2:   0%|          | 1/824 [00:00<01:32,  8.87it/s, training_loss=0.004]
Epoch 2:   0%|          | 1/824 [00:00<01:32,  8.87it/s, training_loss=0.005]
######省略部分过程
Epoch  2:  100%| ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ |  821/824  [01:38<00:00,     8.36it/s,
training_loss=0.299]
Epoch  2:  100%| ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ |  821/824  [01:38<00:00,     8.36it/s,
training_loss=0.003]
Epoch  2:  100%| ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ |  822/824  [01:38<00:00,     8.41it/s,
training_loss=0.003]
Epoch  2:  100%| ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ |  822/824  [01:38<00:00,     8.41it/s,
training_loss=0.002]
Epoch  2:  100%| ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ |  823/824  [01:38<00:00,     8.51it/s,
training_loss=0.002]
Epoch  2:  100%| ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ |  823/824  [01:38<00:00,     8.51it/s,
training_loss=0.003]
Epoch  2:  100%| ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ |  824/824  [01:38<00:00,     8.74it/s,
training loss=0.003]
 50%|██████     | 1/2 [03:24<01:44, 104.72s/it]
Epoch 2
Training loss: 0.2466177608593462
100%|████████████| 2/2 [03:28<00:00, 104.14s/it]
Validation loss: 0.43929754253413067
F1 Score (Weighted): 0.8823813944083021

Epoch 1
Training loss: 0.4581567718019004
Validation loss: 0.3778555450533606
F1 Score (Weighted): 0.8753212258177056

Epoch 2
Training loss: 0.2466177608593462
Validation loss: 0.43929754253413067
F1 Score (Weighted): 0.8823813944083021
```

上述输出结果中，在第一个训练周期后，训练损失持续减小，而验证损失持续增加。这表明模型从第二个周期开始出现过拟合（overfitting）的情况，因此在这种情况下，正确的做法是在第一个周期后停止训练。

过拟合是指模型在训练数据上表现良好，但在未见过的验证数据上表现较差的情况。因为模型在第一个周期已经开始出现过拟合迹象，继续训练可能会导致模型在验证集上的性能下降。停止训练以防止过拟合是一个明智的决策，可以保持模型在验证数据上的泛化性能。此外，可以通过其他技术如早停止（early stopping）来进一步优化模型的训练，以在

适当的时机停止训练并保存性能最佳的模型。

（23）根据前面的输出结果可知，最佳模型是在第一个训练周期（epoch 1）结束时的模型。要加载这个最佳模型以进行预测，可以执行以下操作。

```
model = AutoModelForSequenceClassification.from_pretrained("ProsusAI/finbert",
                                                           num_labels=len(sentiment_dict),
                                                           output_attentions=False,
                                                           output_hidden_states=False)

model.to(device)
model.load_state_dict(torch.load('./finetuned_finBERT_epoch_1.model',
map_location=torch.device('cpu')))
_, predictions, true_vals = evaluate(dataloader_validation)

accuracy_per_class(predictions, true_vals)
```

上述代码加载了之前保存的第一个周期的最佳模型，并对验证集进行情感分类预测。然后，计算并打印了每个情感类别的准确率（accuracy）。代码执行后会输出如下内容。

```
Class: neutral
Accuracy: 385/432

Class: negative
Accuracy: 81/91

Class: positive
Accuracy: 170/204
```

通过执行这段代码，可以获取模型在不同情感类别上的准确率，从而评估模型的性能和情感分类能力，这有助于了解模型在各个类别上的表现如何。