深度学习 NLP 技术和 LSTM 算法模型在量化投资策略中的应用研究

姓名：何雨洵

培养院系：金融学院

指导老师：曹志广

学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品成果。对本人的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

论文作者签名：何雨洵 日期：2021年6月18日

学位论文版权使用授权书

（硕士学位论文用）

本人完全了解上海财经大学关于收集、保存、使用学位论文的规定，即：按照有关要求提交学位论文的印刷本和电子版本。上海财经大学有权保留并向国家有关部门或机构送交本论文的复印件和扫描件，允许论文被查阅和借阅。本人授权上海财经大学可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

论文作者签名：何雨洵 导师签名：曹志广

日期：2021年6月18日 日期：2021年06月18日

摘要

金融量化投资近年来越来越成熟，在国内逐步得到广泛关注，发展成为了投资领域比较热门的研究方向。很多投资机构不再满足于通过宏观基本面分析构建投资组合和交易策略，转变为利用数学、计算机、统计等技术，充分挖掘投资资产的历史数字规律，利用这些数学规律构建量化投资策略。借鉴自欧美国家成熟的量化交易经验，受益于国内金融市场的完善，市场波动率和涨跌幅提高，并且随着衍生品交易起步，量化投资这种新兴的投资策略方式和基于量化的基金产品在国内受到很多投资人青睐。目前，常用的量化投资策略包括多因子股票策略、期货 CTA 策略、对冲套利策略以及高频交易策略等，但是随着市场上的量化投资竞争者不断增多，很多策略都逐渐失效，就需要不断有创新的思路开辟出新的量化投资策略才能获取满足期望的超额收益。针对这种情况，业内已经出现了基于机器学习算法 SVM 和 RNN 等形成的投资策略，期望能够借助机器学习的能力从历史数据中归纳出人不容易直接发现的因子来辅助策略执行，但是由于目前国内诸多交易市场不是真正的有效市场，所以单纯的分析历史行情数据的机器学习策略在应对宏观局势或者突发状况时有比较大的局限性，使得整体策略的效果并不理想。

本文着眼于当前的现状和趋势，提出了一种既考虑到金融市场情绪因素也考虑到金融市场历史交易数据的机器学习量化交易策略。这种策略结合了自然语言处理 NLP 和时间序列处理 LSTM 两种不同的深度学习模型，提取多维度量化因子，构建了一套均衡性基于机器学习的股票量化多因子投资策略。该策略的量化因子主要有三个来源：网络金融舆情文本基于 Bert 自然语言处理深度学习模型做出的情感倾向、股票历史行情基于 LSTM 时间序列分析深度学习模型预测做出的涨跌情况、基于均值回归理论算出的股票价格偏离度比例。综合以上的因子构建出最终的逐日调仓的股票量化交易策略，应用在沪深 300 指数成分股历史行情上进行回测分析和绩效评价，发现该策略效果有明显提升。

本文的通过理论研究和实证分析得出的结论，对于金融机构的股票量化投资策略构建和交易提供了一定的思路参考和论证基础。

关键词：量化投资；机器学习；NLP；LSTM；交易策略

Abstract

In recent years, financial quantitative investment has become more and more mature, and has gradually been widely concerned in China. It has become a hot research direction in the area of investment. Many investment institutions are no longer satisfied with building investment portfolio and trading strategy through macro fundamental analysis, instead they turn to use mathematics, computer, statistics and other technologies, fully mining the historical digital laws of investment assets, and using these mathematical laws to build quantitative investment strategy. Based on the mature experience of quantitative trading in European and American countries, we can also benefit from the improvement of the domestic financial market and the increase of market volatility. With the development of derivatives trading, quantitative investment, the new type investment strategy and quantitative fund product, is more and more favored by many investors in China. At present, the commonly used quantitative investment strategies contain multi factor stock strategy, futures CTA strategy, hedging arbitrage strategy and high-frequency trading strategy. However, with the increasing number of quantitative investment competitors in the market, many strategies are gradually losing effect, so we need to have innovative ideas and open up new quantitative investment strategies to obtain the expected excess returns. In view of this situation, the industry has emerged based on machine learning algorithm SVM and RNN and other forms of investment strategy, expect to be able to use the ability of machine learning from historical data to induce factors that people are not easy to find directly to assist strategy execution, but because many domestic trading markets are not really effective market, the simple analysis of

historical market data of machine learning Learning strategy has great limitations in dealing with the macro situation or unexpected situation, which makes the effect of the overall strategy not ideal.

Focusing on the current situation and trend, this paper proposes a machine learning quantitative trading strategy which takes into account both macro market factors and micro historical data. This strategy combines two different deep learning models of NLP and LSTM, extracts multi-dimensional quantitative factors, and constructs a balanced multi factor investment strategy based on machine learning. The quantitative factors of this strategy mainly come from three sources: the emotional tendency of macro financial public opinion text based on the deep learning model of Bert natural language processing, the rise and fall of stock historical quotation based on the deep learning model of LSTM time series analysis, and the deviation ratio of stock price based on the mean regression theory. Based on the above factors, this paper constructs the final stock quantitative trading strategy of day-to-day position adjustment, and applies it to the back test analysis and performance evaluation of the historical market of Shanghai and Shenzhen 300 index, and finds that the effect of this strategy has been significantly improved.

The conclusion of this paper through theoretical research and empirical analysis provides a certain reference and demonstration basis for the construction and trading of stock quantitative investment strategy of financial institutions.

Keywords: quantitative; machine learning; NLP;LSTM; trading strategy

目录

第一章 引言

1.1 研究目的和意义

量化交易已经成为了金融交易的趋势，机器交易不受人的情绪干扰，能够客观地做出投资决策，交易速度和执行频率上远远超过人工，所以研究量化交易策略就具有实际意义。

在量化投资领域，机器与人类相比不会受到主观情绪的影响，在交易速度和交易频率上都远远胜过人工交易。金融交易数据巨大，传统的金融计量模型在面对多维度海量的大数据进行复杂的分析方面是有无法胜任的，深度学习的手段则弥补了这一点。与图像识别等数据特征不同，金融市场的数据波动性大，具有很强的不确定性，很多交易数据的规律是跟投资人的心理活动相关的，只是单纯将深度学习方法引入金融投资建模，很难达到应有的效果。因此，如何正确地将深度学习方法用到金融投资中，并结合金融市场的数据特征做出机器学习算法的改进，这种研究意义是很大的。

金融新闻情绪分析是属于行为金融学类别，在国内作为研究热点一直存在。虽然国内股市已经发展了二三十年，但由于制度依然不健全，信息有很多不对称，市场中存在较多散户交易，经常追涨杀跌。个股的非理性交易现象比较普遍，普通投资者的投资行为很容易受到宏观新闻媒体影响。进行金融情绪分析研究不仅可以探索行为投资者的投资规律，在当前金融制度改革的大潮中，对于提高金融市场稳定性，降低市场情绪对于市场的冲击，加强金融金融监管能力方面也是非常具有现实意义。

根据国内目前的量化投资研究现状来看，多数量化策略还是停留在多因子选股以及趋势追踪上，还没有充分利用到复杂算法进行建模。深度学习作为当前最先进的人工智能技术，在处理大规模、高纬度的金融数据方面具有得天独厚的优势。从投资者的角度来说，传统的量化投资策略容易受到市场干扰失效，无法准确而紧跟市场地表征交易内在的规律。市场本身存在诸多不确定因素，投资者的心理，宏观的舆论，市场的波动性以及趋势等，需要一个能考虑诸多因素等较为完善的金融预测模型来整体把握市场走向。本文就是根据金融市场情绪和历史数据两方面的因素，

对深度学习算法模型进行了整合，结合了自然语言处理舆情分析和时间序列分析涨跌预测两个方面建立多因子量化策略，对于提供新的量化投资策略逻辑和思路具有比较高的研究价值和现实意义。

本文的创新点主要在于针对现有量化投资策略设计的改良和融合，主要体现在几个方面：

（1）采用网络舆情分析和时间序列分析结合的方式，将深度学习模型完整地应用于股票多因子量化投资策略中，模型设计具有创新性。

（2）基于 Bert 和 LSTM 深度模型，对量化投资模型和策略进行了实现并应用于实际的金融数据进行实证分析和绩效评估，模型实践具有创新性。

1.2 研究背景

世界范围内的各种资本市场在近几十年得到了快速的发展，不仅体现在金融衍生品的种类数量的增加上，也体现在交易投资人数量的增长上，在一定程度上给经典金融理论有效市场假说的理论和实证带来了巨大挑战。根据原有的金融理论，有效市场假说的模型要求投资者是完全理性的，投资人构建投资交易策略基本都是基于宏观基本面信息，认为投资标的的资产价格会在基本面左右浮动。但是从近几十年的机场金融危机事件（1997 年亚洲金危机、2001 年互联网泡沫危机、2008 年金融次贷危机）来看，投资人在面对金融市场进行投资时候往往带有投机性和情绪性，做出非理性的投资举动，这样就给整体金融市场带来了交易噪音。后来的金融研究学者 Black 发现交易市场中的投资者其实不仅包括理性投资者，也包括专门通过噪音交易进行获利的噪音交易者。为了研究这种带有情绪倾向的投资交易行为，学者们提出了各种情绪分析模型，包括 BSV 情绪认知价格模型、DSSW 情绪均衡价格模型、DHS 心里偏差价格模型、BHS 行为资产定价模型。这些模型都从各个方面论证了资产价格和交易市场状况如何受到投资者情绪影响。Shiller 发现有效市场假说很难解释某些时候股票价格存在波动性大的现象，且实证研究结果与事实矛盾。股票价格无法充分反映过往历史行情中的信息，反而很多都是受到公司基本面和市场情绪之类的宏观因素影响。

所以，金融市场情绪分析就出现了，成为了金融学研究的一个重要分支。投资者情绪是千变万化并且难以捕捉的，事实上，早在 1636 年荷兰郁金香狂潮事件中，人们就发现了投资者情绪对于资产价格的影响，投资者的心理会存在从众和恐慌，所以在金融危机来临是，金融市场的急剧变化跟投资者情绪关系密切。在以前，投资者情绪是很难测量的复杂指标，所以相关研究进展缓慢，然而近些年，随着云计算、大数据、机器学习等前沿计算机技术的发展，越来越多的投资者和研究者可以通过公开的新闻媒体报道以及社交网络数据中提取情绪文本信息辅助研究，从推动了金融宏观情绪分析领域的发展。

另一方面，近十年来，人工智能 AI 科技已经成为了全世界范围最炙手可热的科学技术领域，尤其是模拟人工神经元网络的机器学习流派深度学习的兴起，极大地推动了通用人工智能方法的普及。当前深度学习技术的人工智能科技已经广泛应用到语音识别、计算机视觉、自动驾驶和自然语言处理等方面，并取得了重要的研究成果，工业界落地执行的项目也为人类提供了诸多先进的智能化产品，推动了整体科技的进步。因此，基于人工智能技术的量化投资也逐渐成为了比较流行的量化策略构建方法，给量化投资注入新的思路。

深度学习方法是模拟人工神经元网络的，内部有很多参数是通过训练自动调整而来，但是实际上深度学习模型内部是一个黑箱，从数学上证明和解释输入输出的逻辑对应关系是比较困难的，将其应用于金融投资领域在学术界和产业界还是存在一些争议的，需要不断论证是实际应用来验证。预测金融资产的价格理论意义和实践意义是比较大的，现代金融理论认为，过往的金融资产价格数据会影响未来的资产价格，例如 CAPM 模型和 ARMA 模型，这给金融行情预测提供了理论依据。但是，在实际的金融市场中，金融数据量巨大，数据之间的非线性关系是很难用传统金融理论的公式去解释的，而且由于当前的金融市场并非有效市场，传统的金融模型预测的数据会出现较多的欠拟合，泛化能力差。所以，在面对这样的情况时，通过深度学习，运用计算机强大的算力来做资产价格的预测就会成为一种可能和新的思路。学术界和产业界当前都投入了大量资源在探索如何将深度学习完美地应用于金融研究中，为金融研究打开新的视角。

1.3 文献综述

有研究人员运用多元逐步回归法得出（汪洋，2010）上市公司财务状况的好坏将会直接影响投资者的收益率，财务状况好的公司倾向 于给投资者带来更高的收益和更低的风险。（江方敏，2013)将 11 个量化选股 因子纳入量化选股模型中进行分析，最终发现 11 个量化选股因子中只有 4 个因 子是最有效的量化选股因子，分别是市盈率、市营率、市净率和现金流增率。（蔡立瑞，2017)基于文本分析的投资组合与量化选股的框架构建多元回归模型，得出华夏银行 2014 年 1 月至 2015 年 4 月股票收益率对模型中的市场投资组合风险溢价因子、账面市值比因子是敏感的，即这两个因子可以部分解释华夏银行的收益率变动，而市值因子的系数不显著，解释能力不够强。（张静、黄东军，2010)选取 EMA、DIFF、 DEA 和 MACD 做为选股的依据，结果发现，选出来的股票组合能获取超越市场的收益。（聂洣佳，2016)设计出了以 BOLL、MACD 和 RSI 三个技术面指标为选股条件的选股器，其思路是通过这三个指标选取大量股票进行投资，来实现收益量的提升。

（Fischer，2017)研究了深度学习在金融市场的应用，在 Krauss 的研究基础上也利用 1992 年至 2015 年 S&P500 的股票数据进行预测，在样本外预测获得了 0.46%的日收益率和 5.8 的夏普比率，实验结果显示，基于 LSTM 构建的股票交易策略，回测效果要好于随机森林、深度神经网络和逻辑回归。（Ryo Akita，2016)构造出了一种利用 LSTM 网络预测金融时间序列数据的深度学习模型。该方法利用深度学习模型分析公司开盘价与报纸上对于该公司的过去事件的时间效应，投资者根据改模型的分析结果进行投资，该方法对在东京证券交易所上市的五十家公司的实际数据的分析结果表明了深度学习方法在预测股票价格方面的的有效性。（Wei，2017) 得出了一种新的深度学习框架，并首次将小波变换、堆叠和长短记忆神经网络相结合进行股票价格预测。该深度学习模型分散个阶段，首先利用小波变换对股票价格的时间序列进行降噪处理，然后利用堆叠进行分层特征取，最后将处理好的特征输入 LSTM 网络进行次日收盘价预测，并利用了 6 个股票市场指数期货来 验证模型的性能，研究结果表明，该模型在预测精度和盈利能力方面均优于同类模型。

1.4 研究方法

本文采取理论研究与实证研究同步推进，理论研究与实际交易策略回测相互验证整合的思路。课题研究包含以下方法:

（1）文献阅读与数据理论。阅读学习目前最新的相关的研究成果，对比分析不同的研究方法、网络模型的异同，归纳研究现状。对量化投资和机器学习的最新研究成果的学习和在金融市场业务应用，研究多因子量化交易策略的理论与方法，构建基于机器学习方法的多因子股票投资策略

（2）实证分析。以目前股票市场的股价数据和网络社区的金融评论数据作为训练集和测试集，构建量化策略模型，通过多种方法对数据进行处理，验证交易策略的效性进行。

第二章 基础理论和技术

2.1 行为金融学

2.1.1 行为金融学概述

行为金融学从微观个体行为以及产生这种行为的心理等动因来解释、研究和预测金融市场的发展。这一研究视角通过分析金融市场主体在市场行为中的偏差和反常，来寻求不同市场主体在不同环境下的经营理念及决策行为特征，力求建立一种能正确反映市场主体实际决策行为和市场运行状况的描述性模型。

行为金融学虽然试图深挖金融市场运作背后的奥秘，但并不系统也不透彻，因此现今成型的行为金融学模型还不多，研究的重点还停留在对市场异常和认知偏差的定性描述和历史观察上，以及鉴别可能对金融市场行为有系统影响的行为决策属性。行为金融学是一种研究金融市场的新思路，至少它试图解决传统研究范式中不能解决的那一部分。广义上说，行为金融学认为代理人并非完全理性，通过这种假设可以更好的理解某些异常金融现象。更具体而言，行为金融学主要分析在放松“理性”的限制条件后金融市场的行为。换言之，在某些行为金融模型中，代理人无法正确地更新自己的理念；而在另外一些模型中，代理人恰当地考虑了 Bayes 理论，但是其决策不一定是普遍认同。

2.1.2 行为金融学起源

早在半个世纪前，爱德华就将决策的制定引入心理学的研究领域，并勾画了未来研究的蓝图。但认知心理学的理论研究直到卡内曼和屠夫斯基发表他们在判断和决策课题上的研究成果才取得巨大的突破。在介绍卡内曼的贡献之前，我们将首先介绍经济学和心理学在关于决策制定的理论假设方面的一些本质的区别。

行为金融学是金融学、心理学、行为学、社会学等学科相交叉的边缘学科，力图揭示金融市场的非理性行为和决策规律。行为金融理论认为，证券的市场价格并不只由证券内在价值所决定，还在很大程度上受到投资者主体行为的影响,即投资者心理与行为对证券市场的价格决定及其变动具有重大影响。它是和有效市场假说(efficient market hypothesis,EMH)相对应的一种学说，主要内容可分为套利限制（limits of arbitrage）和心理学两部分。

20 世纪 50 年代,冯·纽曼和摩根斯坦(VonNeumannandMorgenstern)在公理化假设的基础上建立了不确定条件下对理性人选择进行分析的框架,即期望效用函数理论。阿罗和德布鲁(ArrowandDebreu)后来发展并完善了一般均衡理论,成为经济学分析的基础,从而建立了现代经济学统一的分析范式。这个范式也成为现代金融学分析理性人决策的基础。1952 年马克威茨(Markowitz)发表了著名的论文“portfolioselection”,建立了现代资产组合理论,标志着现代金融学的诞生。此后莫迪戈里安尼和米勒建立了 MM 定理,开创了公司金融学,成为现代金融学的一个重要分支。20 世纪 60 年代夏普和林特纳等建立并扩展了资本资产定价模型(CAPM)。20 世纪 70 年代罗斯(Ross)基于无套利原理建立了更具一般性的套利定价理论(APT)。20 世纪 70 年代法马(Fama)对有效市场假说(EMH)进行了正式表述,布莱克、斯科尔斯和莫顿建立了期权定价模型(OPM),至此,现代金融学已经成为一门逻辑严密的具有统一分析框架的学科。

2.2 量化投资

2.2.1 量化投资概述

量化投资是一种借助数学、统计分析、计算机等科技手段进行的投资方式，一般是将金融数据进行统计分析建模后使用计算机程序按照量化策略逻辑执行自动化交易以获取超额收益。量化投资有几个方面的特点：

（1）纪律性

量化投资是通过设定好的策略模型进行程序化交易，其内在交易逻辑是提前设定好的，所以在交易过程中全自动化，人工不干预，能够较少地受投资者情绪影响。

（2）及时性

量化投资通过计算机的技术可以很快捷地捕捉金融市场信息，快速决定入场的交易时机。从行情、策略、交易执行等各个过程都实现了自动化，在不断采集金融数据的同时还能实时优化策略自身的参数。

（3）系统性

量化投资涉及到一整套金融解决方案，包括行情、交易、风控、策略以及回测和绩效分析多个模块。制定交易策略也需要一整套完善的因子挖掘、策略实现、策略回测到实盘运行的过程。

（4）分散化

量化投资可以很好的保证分散化投资，鸡蛋不放在一个篮子里。量化投资策略会根据因子进行选股，分成不同的投资组合，尽量追求低风险高收益。

（5）准确性

量化投资实在以客观科学的方式寻找投资价值，借助计算机的准确运算能力，能够全方位地计算买入卖出价格，数量等，并且使得交易时机更加精确。

与量化投资相对应是传统投资，传统投资主要依靠投资者对于宏观和基本面信息的把我，带有比较明显的主观性，并且人工处理信息的速度是远远不及量化投资的。传统投资是通过价值评估来制定投资决策，而量化投资择时通过科学计算的方法进行建模来构建投资策略。从投资周期层面来说，量化投资更侧重于短期投资，

传统投资则侧重于长期投资。较传统投资而言，量化投资可以用过风控模型来很好地控制投资风险，包括投资组合风险和交易风险。

2.2.2 量化投资策略概况

量化投资策略的种类和流派众多，按照交易产品、盈利模式、策略思路、交易速度等多个方向，可以分成几类。

按照交易产品分类，可以分为股票多策略、期货 CTA 策略、期权策略。其中 CTA 策略指的是交易商品期货、股指期货等品种的策略。股票策略就是交易股票的量化策略。

按照盈利模式来分，可以分为套利策略、对冲策略、单边策略。套利策略又包括跨期套利和跨市套利以及跨品种套利，这种策略一般风险较低。对冲策略主要是指通过多空反向去持有不同品种或组合的仓位实现控制风控的策略。单边策略是投资者在结合价值分析或历史数据的基础上，对金融产品进行单边买入或卖出实现盈利的策略。

按照策略信号划分，可以分为均值回归策略、动量策略、因子策略等。不同的信号决定了制定交易决策的来源不同，采用的技术手段也会不同。

按照交易速度区分，量化投资大体上可以分为中低频量化和高频量化。中低频量化一般不要求订单执行速度，或者对于订单执行的延迟容忍度较高，比如股票相关的策略，在国内由于交易所种种限制，就属于中低频策略。而高频量化策略一般多见于低延时高频交易，主要交易期权期货等品种，要求订单延时在微秒级，并且频繁报撤单操作，每天清理头寸，调仓频率也非常高，如果要实现真正的高频量化策略，对于计算机交易程序的技术能力要求比较高。

按照计算机程序智能化手段划分，可以划分为传统量化策略和人工智能量化策略。传统量化策略就是通过认为设定的统计模型区制定交易决策，人工智能量化策略，择时在策略中采用了机器学习的手段去进行因子的挖掘和模型参数的制定，这种策略能够探索出人工不容易发现的金融投资因素。

2.3 深度学习

2.3.1 机器学习概念

机器学习(Machine Learning, ML)是一种计算机技术，一般是指通过一定的预定义算法根据大规模数据训练调整参数并进行数据预测的技术。机器学习涉及到数据挖掘、统计学、计算机视觉和文本处理相关学科，主要是研究如何用计算机模拟人类的思维和逻辑去完成任务。机器学习和人类的学习过程是相似的，人类的学习是通过不断记忆、归纳、总结从而形成经验或者技能，机器也可以模仿这种过程，根据历史数据去训练、归纳、总结形成机器学习模型的算法参数，来达到目的。人类可以根据历史的经验去预测未来的发展走向，机器也可以根据过往数据预测未来的发展趋势。

机器学习分成有监督学习和无监督学习和强化学习。大部分机器学习任务需要通过历史数据和标签进行训练，这种是有监督学习。无监督学习可以在没有预定义标签的海量数据中自动寻找出规律，发现数据中的嵌在关系。强化学习则介于有监督学习和无监督学习两者之间，没有明确的数据标签，但是每次学习都可以获得相应的反馈调整参数。

有监督学习的常用算法包括：决策树（Decision Trees）、朴素贝叶斯分类(Naive Bayesian classification)、最小二乘法（Ordinary Least Squares Regression）、逻辑回归(Logistic Regression)、支持向量机（Support Vector Machine，SVM）、集成方法（Ensemble methods）。无监督学习常用算法包括：聚类算法（Clustering Algorithms）、主成分分析（Principal Component Analysis，PCA）、奇异值分解（Singular Value Decomposition，SVD）、独立成分分析（Independent Component Analysis，ICA）。强化学习常用算法包括：Q-learning、Policy Gradient、Proximal Policy Optimization(PPO)。不同的机器学习算法原理不同，但是在模型构建和训练过程使用的损失函数、梯度下降等方法很多都是公共的。

2.3.2 神经网络理论

人工智能发展了几十年，在早期的时候，人工智能技术还停留在知识工程和决策树分析这种浅层结构上，一般只包含很少的非线性变换。广义上来说，机器学习的常用算法，比如逻辑回归（LR）、支撑向量机（SVM）、多层感知机（MPL）等都是浅层结构。多年的研究和实践表明，浅层结构的机器学习方法能够解决简单的分类和回归问题，但是有诸多的制约条件，建模能力和表示能力是不足的。浅层结构的机器学习在处理人工智能领域的复杂任务（语音识别、图像识别、机器翻译、自动驾驶和自然语言处理等）时比较困难且效果较差。

类比人的信息处理机制，浅层结构的机器学习其实跟人类的信息处理（听觉、视觉、嗅觉）有很大区别，人类处理信息的过程涉及到接触神经元细胞到复杂的多层次感官细胞的处理，从复杂的信息转换为内部信号最终传输到大脑综合处理。研究发现，人类的语言输入和感知系统是具有负责的网络结构的，视觉系统则具有分层结构。这些特征是值得机器学习模仿的。

深度学习是机器学习的一种分支，它正式起源于人工神经网络的研究。人工神经网络（Artificial Neural Network,ANN），是机器学习算法中最接近生物大脑构造的数学模型。人工神经网络模拟生物的神经网络结构功能，通过大量的神经元节点联结，每个神经元成为 MP 模型，将大脑处理神经信号的方式用数学公式表示出来，包括输入输出函数和激活函数。将大量的神经元节点按照分层连接，并且配置多维度参数，共同组成了复杂的人工神经元网络。

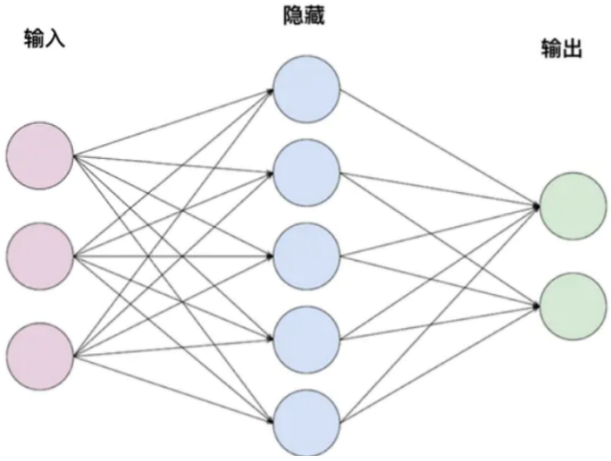

图 1：人工神经网络

2.3.3 深度学习理论

深度学习是依托于人工神经网络的新型机器学习方法，顾名思义，深度指的是多层神经网络。这些深层神经网络搭配上复杂的分层参数共同构成了深度学习模型，便于处理大量非线性信息。深度学习训练容易遇到欠拟合跟过拟合问题。为了解决拟合问题，研究者发现训练集的误差会根据时间的推移不断变低，但是误差会提高。所以，学术界发现了一种思路，就是循环地去根据已训练的参数误差调整参数本身，直到训练终止。

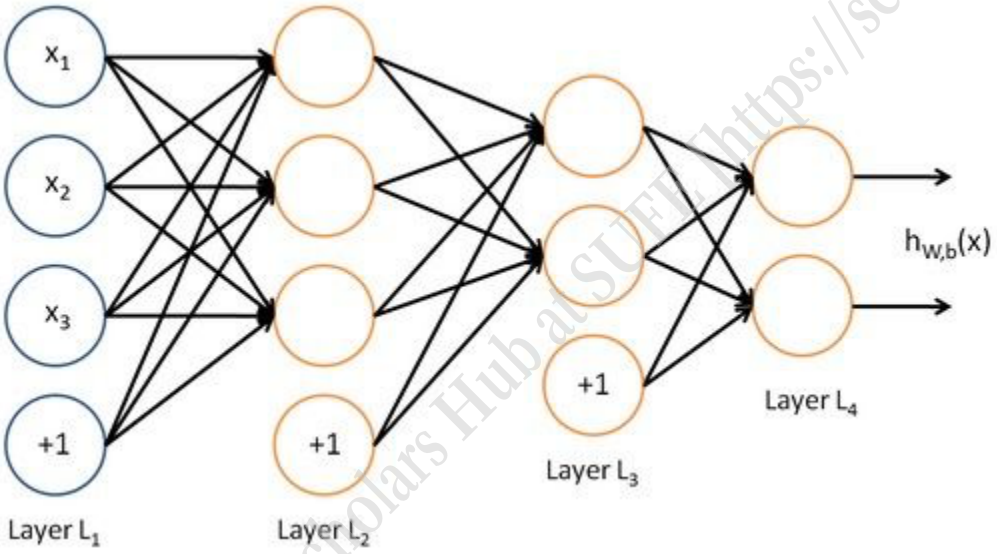

图 2：深度学习模型

早期在 1982 年的时候，杰弗里辛顿提出了多层感知机反向传播算法（BP 算法），这基本上可以称为深度学习的雏形。BP 算法算法添加了反向传播机制，可以在训练中调整神经元的权重和阈值参数，知道误差在合理范围内。但是 BP 算法只是解决了非线性分类问题，在更复杂的机器学习任务中无法胜任。另一方面，在上世纪八十年代的计算机硬件能力是非常有限的，相关软件业非常不完善，导致整体的运算能力跟不上，再加上 BP 网络在神经网络层数增加时会出现梯度消失问题，这使得在上世纪后期基于神经网络的机器学习发展收到很大制约。深度学习的概念是有 Hinton 在 2006 年提出的，早期是为了解决深度置信网络问题，为了深层结构优化而服务的。深度学习的好处是可以用无监督式的手段替代人工获取标签特征，模拟人脑的原理解释图形、语音、文本等数据输入。深度学习采用分层抽象结构，

一般包括输入层、隐层和输出层，不同层次的参数维度各不相同，基于高层抽象的算法完成数据推断或预测任务。随着深度学习的快速发展，到了 2014 年，facebook 公司深度学习项目 DeepFace 已经可以把人脸识别准确度提高到 97%以上，而 2016 年，谷歌公司的深度学习项目 AlphaGo 围棋人工智能机器人更是以 4:1 的大比分战胜了国际围棋冠军，从此以后深度学习进入了新世纪的火热爆发期，在众多领域的表现超越人类。

深度学习发展以来，历练出了很多经典算法，比如卷积神经网络（CNN）和循环神经网络（CNN）以及便于处理时间序列信息的长短期记忆网络（LSTM）。至今有多款专用于深度学习的程序框架，比如 google 公司的 tensorflow 和 facebook 公司的 pytorch 这两种在工业界和学术界有代表性的工具框架。深度学习广泛应用于计算机视觉、自然语言处理、语音识别等领域，为不同行业注入新的活力和科技创新。

第三章 基于自然语言处理的深度学习量化因子模型研究

本章的主要内容是根据网络爬虫采集的互联网金融股票评论数据，构建和训练自然语言处理(NLP)文本情绪分析深度学习模型，基于模型设计出量化交易策略能够使用的金融舆情因子。

3.1 自然语言处理 NLP

3.1.1 NLP 概论

自然语言处理（Natural Language Processing, NLP）是计算机人工智能领域的一个重要研究分支，主要研究的是通过计算机对自然语言进行分析、理解、识别，广泛应用于机器翻译、自动摘要、文本分类、智能问答等多项实用性场景当中。

自然语言处理包括了自然语言理解和自然语言生成两个方面，自然语言处理经过长时间的发展，已经形成了很多常用研究分析技术。

（1）词干提取

词干提取的过程是通过去除衍生和变化形态，将词语抽取为词干原型的过程。词干提取的目的是将多样化的词语转变为公共的词干，生成的词干又可以用来扩充词典的词目。

（2）分词

分词一般指的是中文分词，将连续的一段语句按照词语的语义进行切割组合成词语序列的过程，同样一段文字在经过中文分词后可以有多种结果。

（3）词形还原

词形还原可以将一组词语还原成词源或词典的词目，还原的过程要考虑到词语在句中的实际含义以及相邻语句的语义。

（4）词向量化

词向量话指的是把词语或者字用数字索引标识的方法。由于计算机无法处理自然语言文字，必须将文字转成数字索引值，才能接入实际的研究分析。通过词向量化，可以将一句话转成定微的向量数组表示。

（5）词性标注

词性标注是对句子中的词语根据词性分类标注为名词、动词、形容词、副词的过程。

经过这种归类后，便于建模分析时候根据类别区分。

（6）命名实体识别

命名实体识别具体是识别句子中有特定意义的实体词语，然后将其区分为人名，机构名，日期，地名，时间等类别的任务。

（7）情感分析

情感分析实际上是一种带有人为主观情绪的分析，借助自然语言文本处理技术识别相应的文本中带有的语义情感，根据语句表达的态度或情绪进行情感分类，识别出文本中带有的积极或消极、喜怒哀乐等自然情感。

（8）语义文本相似度分析

语义文本相似度分析指的是通过量化的方式计算两段不同文本的相似度，便于分析其内在关联。

（9）语言识别

语言识别指的是将不同语言的文本区分出来，比如区分英文和法语，主要是利用语法区别和语言属性来达到目的。

（10）文本摘要

文本摘要是通过计算机算法提取一篇文章的核心内容压缩成一段简略文本概括内容的流程，可用于搜索和文献分析。

自然语言处理涉及到语法、语义、语音等多维度分析，基本任务在于词典、词频、上下文语义分析等处理方式。如今，自然语言处理（NLP）应用无处不在，这种使用率的广泛使用得益于大规模基础公共数据集预训练模型并基于这之上的迁移学习，所以挑选和搭建一个优秀完善的 NLP 深度学习模型来完成文本分析任务就变得尤为重要。在本文中，拟采用 BERT 模型来参与金融舆情文本分析，该模型的原理如下文所述。

3.1.2 BERT 模型介绍

谷歌的 AI 团队于 2018 年发布了专门用于自然语言处理的 BERT 模型，这种新型的 NLP 深度学习训练模型横空出世之后，就在世界级的多项机器阅读理解等顶级水平测试中表现优异，在全部多个衡量指标上的识别率超越人类，而且在超过 11 种不同的 NLP 基准测试中成绩斐然，包括 GLUE 基准达到 80.4%，MultiNLI 基准达到 86.7%。可以说 BERT 模型的到来堪称 NLP 技术的里程碑，给近些年的 NLP 领域带来重要进展。

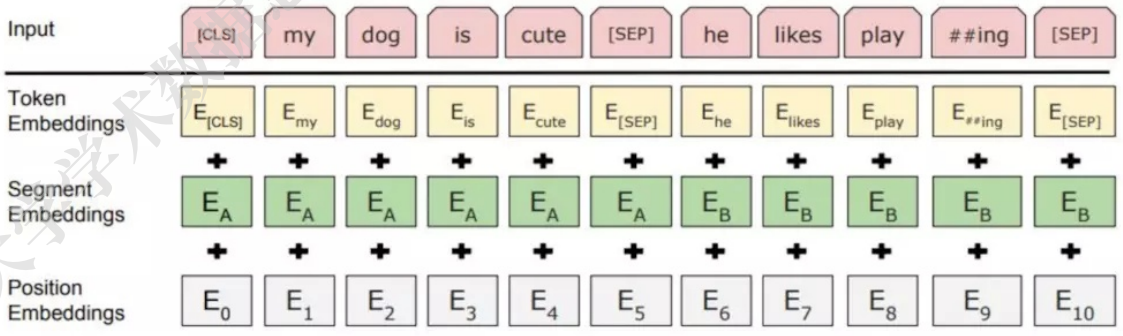

图 3：bert 模型词嵌入

BERT 的新语言标识模型，是用 Transformer 的双向编码器构成的。与其他语言模型不同，BERT 可以调节所有层的上下文来，通过额外的输入层微调深度等参数，非常适合于进行语言推理。

BERT 自然语言深度学习模型由下列特点：

1） 模型深度层次较深，总共有 12 层，但不宽，中间隐层只有 1024 层，相比较以前的主流 NLP 领域 transformer 模型中间层 20148 层少了一半。这基本也标示着计算机人工智能深度学习领域的一个趋势，也就是深而窄的模型比浅且宽的模型表现要更好。

2） 采用了遮罩语言层（Masked Language Model），既利用了左侧的词语，也利用了右侧的词语，这种方式在之前的 ELMO 模型中也出现过，不是 BERT 原创。

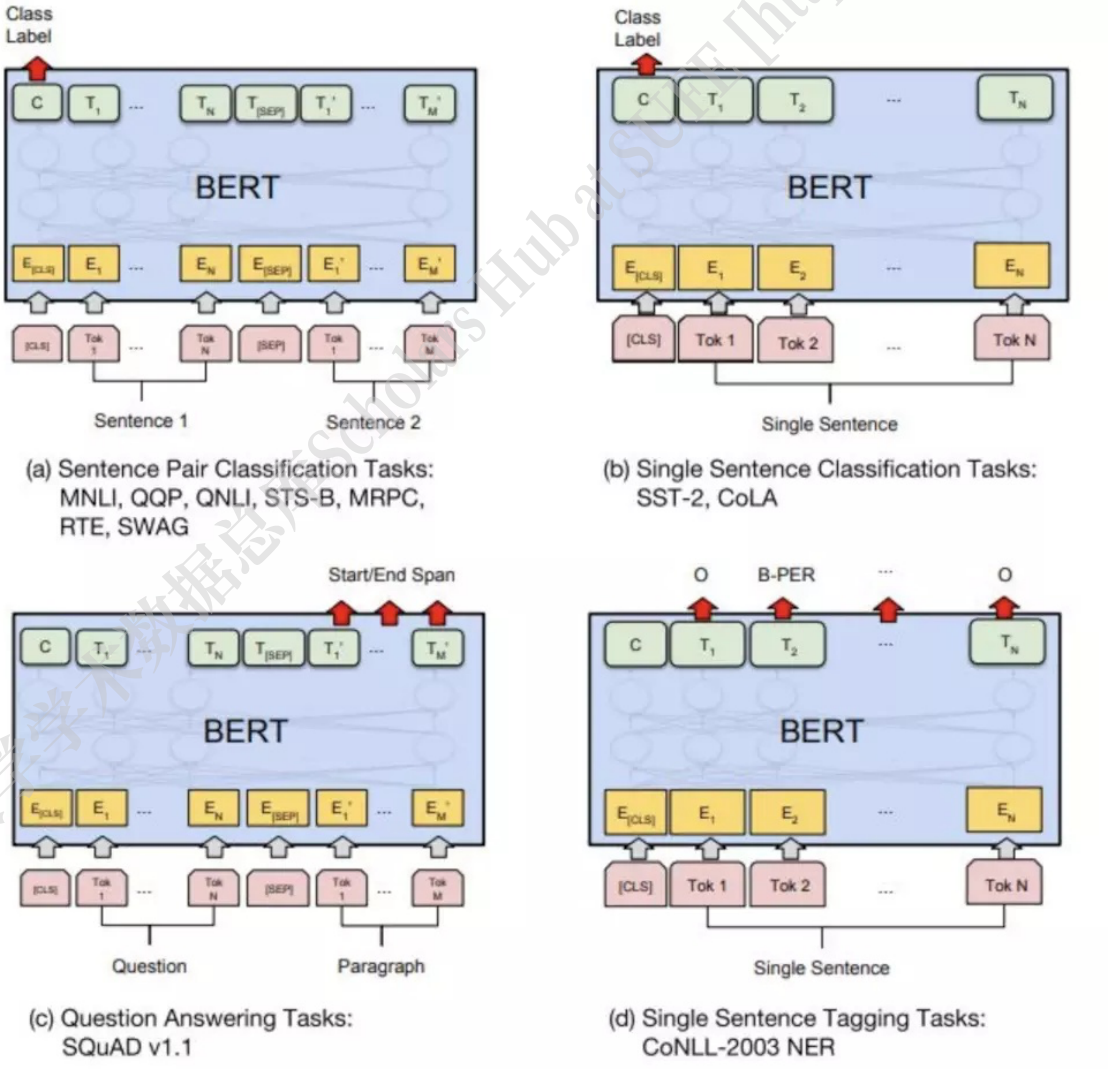

图 4：Bert 多项任务的结果

BERT 提出了一种新的预训练目标，也就是遮罩语言模型（MLM），能够有效克服单向性局限。MLM 输入中的 token，可以仅基于遮罩词的上下文语义场和语境就推理出其原始词汇。

总结起来，BERT 模型就是一个依赖超大规模数据和计算机算力进行预训练而成的自然语言模型，能够提供精确度高的、性能好的文本表征能力。

3.2 金融舆情文本数据采集

3.2.1 网络爬虫

本章的金融舆情文本数据是从互联网股评社区东方财富网获取的。东方财富网是国内比较主流的金融交易投资和财经信息传播的网络平台，涵盖的内容比较广，包括财经资讯以及股票、期货期权、外汇等多品种的行情数据。对于每个证券都有属于整个证券相关信息的讨论区，东方财富每个证券讨论区的评论内容基本代表着最广大的市场上对于股市情况的宏观评价。

股票评论文本内容存在于互联网上，数量很大，用人工录入的方式是不可能，不仅耗费时间巨大而且容易出错。所以，这里采用 python 编写的爬虫程序对东方财富网股票评论社区进行自动下载抓取，然后解析出关键字段信息数据以结构化的形式存储，为后续建模分析调用做准备。

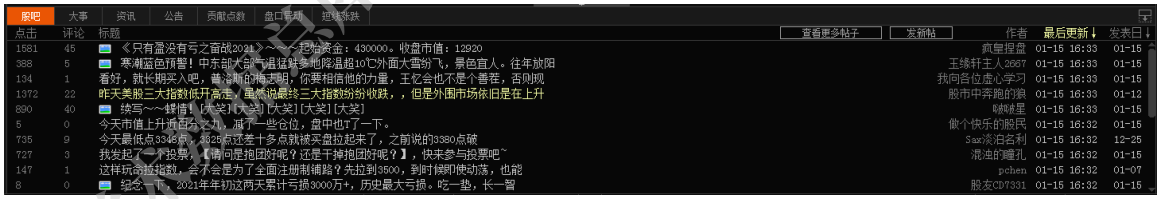

图 5：股票评论区

图：爬虫流程

部分爬虫程序片段如下：

```
for page_id in range(1, max_page):
            page_url = "http://" + host_url + "/list," + market +
instcode + ",f_" + str(page_id) + ".html"
```

```
            # use reandom delay to avoid website protect
            sleep_time = delayRandomTime(0, 1)
            print ("sleep for %f s" % sleep_time)

            html = getHtml(page_url)
            soup = BeautifulSoup(html, "html.parser")
            raw_comment_links = soup.find_all("div", {"class":
"articleh"})

            print ("--- current page_id:", page_id)
            print ("--- current page_url:", page_url)

            for raw_comment_link in raw_comment_links:
                # parse every comment overview item
                comment_link_content = raw_comment_link.find("span",
{"class": "l3 a3"}).find("a")
                comment_url = comment_link_content["href"]
                if "//" in comment_url:
                    comment_url = "http:" + comment_url
                else:
                    comment_url = "http://" + host_url + comment_url
```

本文金融舆情数据爬虫程序的流程如图所示：首先，用 python 调用财经数据包 akshare 获取沪深 300 指数以及成分股的股票代码。然后，根据每个股票代码，生成个股评论区 url，用 python 程序模拟浏览器从东方财富网请求对应的股票评论区 url 列表。根据响应得到的 html 标签以及既定的规则解析 html 的字符内容，提取股票评论、点击量、回复量、日期时间等关键信息。最后，使用 python 程序将提取得到的文本数据存储到 csv 文件。

3.2.2 结构化预处理

使用前述的爬虫程序从东方财富网共获取了沪深 300 成分股 2020 年 1 月 1 日到 2020 年 12 月 31 日股评数据。爬虫程序抓取下载后根据每个股票按照时间、评论内容、点击量、回复量进行结构化存储。如下表所示，从结构化 csv 文件中读取的美的集团股票（000333.SZ）的部分股评数据。

表 1：美的集团部分股评

security	comment	read	reply	date	time

000333.SZ	美的真的很强势	269	0	20201021	09:36
000333.SZ	恭喜美的盘中 78.05、收盘新高 77.50 皆创新高	337	3	20201021	15:36
000333.SZ	好戏在后头	313	2	20201021	15:25
000333.SZ	选美的的原因	280	5	20201231	09:33
000333.SZ	得了空头彻底被打爆了	199	1	20201231	09:31

3.3 构建 NLP 模型量化因子

本部分使用有监督学习的深度学习方式构建金融舆情文本情感分类模型。先将原始股评数据进行预处理，包括人工标注、中文分词、词向量化、特征选取，然后将数据分位训练集和测试集，基于 BERT 构造的完整 NLP 处理模型训练得到最终的股评情感分类器。最后，依托于训练好的模型设计用于量化交易策略的网络舆情因子。

3.3.1 人工标注

前面介绍了常用自然语言处理技术中提到了人工标注技术，目的是预先准备好有监督机器学习所需的学习标签。首先需要把整体的数据集切割成训练集和测试集，然后用人工标注的办法把训练集和测试集都标注上标签，这里数据集本身就作为模型输入，标签就是模型输出。

为了接下来的进一步研究，就要对爬虫采集到并结构化预处理后到股评文本数据做标签标注。基本的步骤是，针对每一条股评，看涨的文本标注为 positive，看跌或者横盘的文本标注为 negative。

考虑到数据量巨大，全部由单人手工标注耗时较大并且较困难，同时考虑到位数数据的客观性，从沪深 300 成分股中随机选取不同股票多评论文本共 8000 多条股评文本数据分发给互联网众包平台，由多位第三方人士进行集中标注完成。部分标注结果如下表所示：

表 2：部分股评标签

股评文本	标签
涨得凶猛	positive
强势突破，明天还有的涨的	positive
明天接着买	positive
跳水鸭，不然打屁股[生气]	negative

强弩之末了？	negative
跳水王称号今天送给你![加油]	negative
后悔没上车，太牛了	positive
垃圾刚进就大跌	negative

3.3.2 特征选取

在使用深度学习自然语言处理进行文本分类训练前，需要把所有股评数据转化为程序可以识别的表示，并从中提取特征。例如：“今天要见顶了，明天要回调”（标签：negative）这句话是依据中文汉字表示，计算机程序是无法直接识别并从中得到有用信息的，必须先转为程序能够识别的数值形式，然后从这些数值纬度中抽象出程序识别的数据特征，参与模型训练。

本章所采用的方法是，将每句评论中的词进行向量化（word2vec），然后将整句话转换成词向量的数组序列，而标签则用 1 和 0 数值来分别表示情绪倾向，再将词向量后的一维数组每个元素值作为模型输入特征值。

比如，前面的那句“今天要见顶了，明天要回调”采用中文字典进行词向量化后的表示就是 [101, 791, 1921, 6206, 6224, 7553, 749, 8024, 3209, 1921, 6206, 1726, 6444, 102]这样的一组词向量数组，数组中每个词的索引值就作为这句股评的模型输入特征值，其标签则表示为 1，作为输出值。

3.3.3 模型构建

做完前面的准备工作之后，就可以构建完整 NLP 模型来挖掘出金融舆情文本情感的规律。采用 pytorch 机器学习框架的 BERT 预训练层叠加上线性层形成深度学习模型，模型构建主要分为两个阶段：模型训练、模型预测。

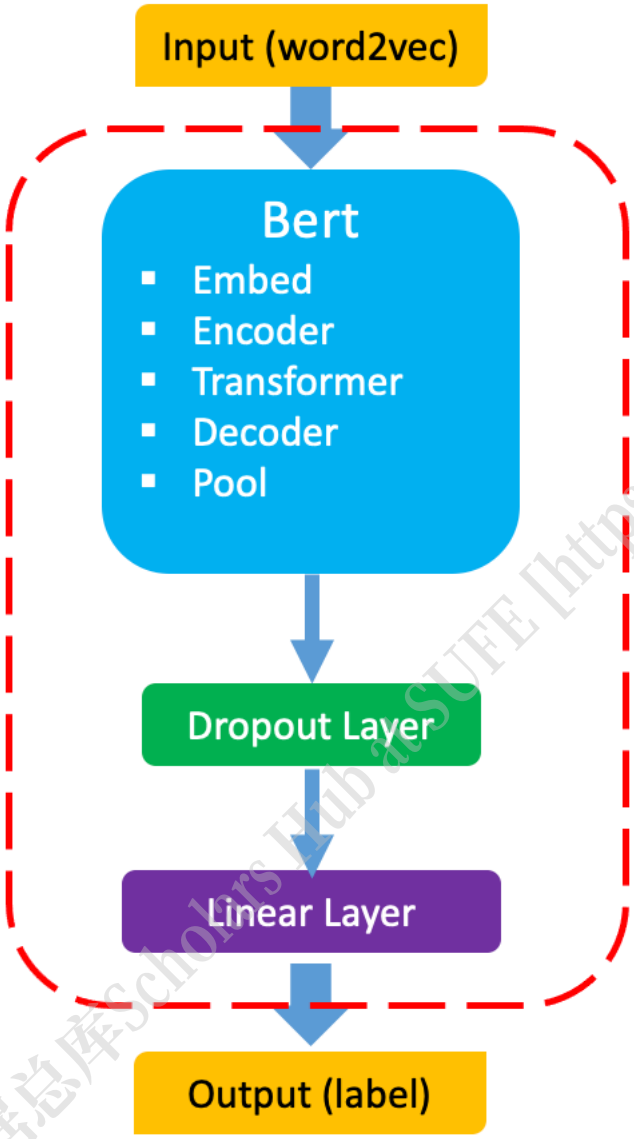

图 6：NLP 深度学习模型结构

（1）模型训练阶段

将人工标注过的 8000 多条股评文本数据划分为训练集和测试集，都进行词向量化、纬度补齐。导入到 python 编写的 NLP 深度学习模型进行训练，直到误差值收敛和准确度攀升到合理范围内结束，然后导出训练好的模型。

（2）模型预测

使用训练好的模型，将完整的所有爬虫采集的股评数据导入，输入到模型中得出输出值进行预测。这样就能得到各股票每一条股票评论代表的情绪是看涨 postivie 还是看跌 negative。

3.3.4 因子设计

深度学习 NLP 模型分析金融舆情最终是为量化投资策略设计量化因子服务的。此处，将每只股票每个交易日当天预测出来的股评情绪按照 positive 和 negative 条目数占当天评论总数的比例作为量化因子，计算公式为：

$$NLP\_SIGNAL = \frac{N_{positive}}{N_{total}}$$

根据以上计算出来的金融舆情因子值，可以作为后续量化投资策略的交易信号组成元素，并参与进一步分析。

第四章 基于时间序列的深度学习量化因子模型研究

本章的主要内容是根据股票历史日级别行情数据，构建和训练时间序列分析专用的 LSTM 深度学习模型，基于模型设计出量化交易策略能够使用的涨跌预测因子。

4.1 时间序列分析

4.1.1 时间序列分析概论

时间序列（Timeseries）指的是按时间先后顺序组成的数值序列。时间序列一般是现实中真实的一组序列数据，不是理论实验中得来的中间数据，反映某一现象的统计指标，可以说时间序列的背后会反映某真实现象的按时间变化的量化规律。

对于时间序列分析的建模遵循一般的步骤。首先，要用观测调查或统计抽样的办法取得时间序列数据，在根据数据做出相关性分析，算出自相关函数，做出相关图。相关图展示数据变化的趋势，还能显示拐点和跳点。跳点指的是与常规数据差距很大的数据，如果跳点是真实存在的值，则时间序列分析时一并考虑进去，如果是异常值，则应该按照期望值做调整或剔除。拐点则是指上升趋势或者下降趋势急剧转变的分界点，如果时间序列存在拐点，那么在建模时需要用一定的拟合算法取拟合拐点前后的序列。时间序列分析中经常涉及到各种各样的曲线拟合方法，用不同的拟合算法取补足或修订时间序列中的观测数据。

时间序列分析属于一种定量预测方法，包括统计分析模型的建立和推断，和关于时间序列的预测和滤波等。传统的统计分析都基本都假定数据序列具有独立性，但是时间序列分析更侧重数据本身的前后相关依赖关系。其实，时间序列分析也可以看成是对于离散值的随机过程分析，基本原理是：关注事物发展的延续性，根据过去的事物情况预测将来的事物发展趋势；考虑事物变化的随机性，采用随机过程找到事物发展的内在规律。

时间序列数据，有多种表现特征形式，不同的形式又有对应的特定分析方法，如下：

（1）长期趋势变化

在某种因素的影响下，时间序列数据变化呈现出一种长期的发展倾向，按照确定的规则持续变化，这种情况下常用分析方法有一栋平均、指数平均、曲线拟合等。

（2）周期性趋势变化

在周期行因素等影响下，时间序列数据变化呈现出周期行变化或循环，这种情况下采用的分析方法是周期性指数法。

（3）随机性趋势变化

由许多不确定因素影响，时间序列数据变化没有确定的规律，呈现随机性变化。采用的分析方法是随机过程分析。

对于本章的研究对象来说，金融行情数据就是一种时间序列，但其变化规律同时具有长期趋势、周期性趋势以及随机性趋势多种特征，所以有多种方式去进行时间序列分析处理。常用金融收益率时间序列分析模型 ARMA 就其中之一。自回归滑动平均模型（Autoregressive moving average model，ARMA）将收益率指标随时间推移的值看作随机序列，同时这组随机数据所具有前后相关关系又能反映原始数据在时间上的延续性。但是由于 ARMA 对于市场有效性等要求的条件过于理想化，所以对于真实的金融行情数据分析有局限性，没有太多实际意义。

4.1.2 LSTM 模型原理

循环神经网络（Recurrent Neural Network，RNN）这种深度学习神经网络能够很好地处理时间序列数据，与其他神经网络相比，它存储了上下文并能在内部建

立前后相关性，比如，它能够很好地处理根据前几个词语推断后几个词的文本推断任务，也能处理诸如预测空气指数这种周期性数据任务。

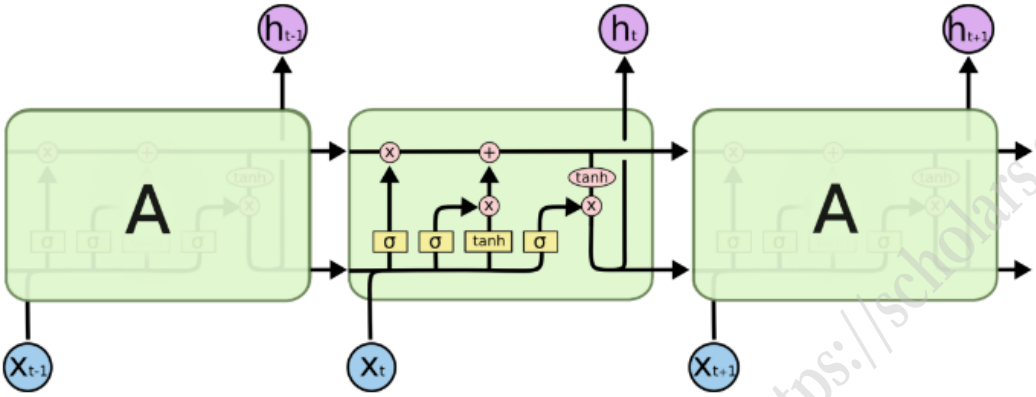

图 7：LSTM 内部结构

而长短期记忆网络（Long short-term memory，LSTM）就是一种非常特殊的 RNN，LSTM 相比普通的 RNN，更好地解决了时间序列训练中的梯度消失或爆炸问题，在很多深度学习任务应用中，LSTM 可以获得更好的表现。

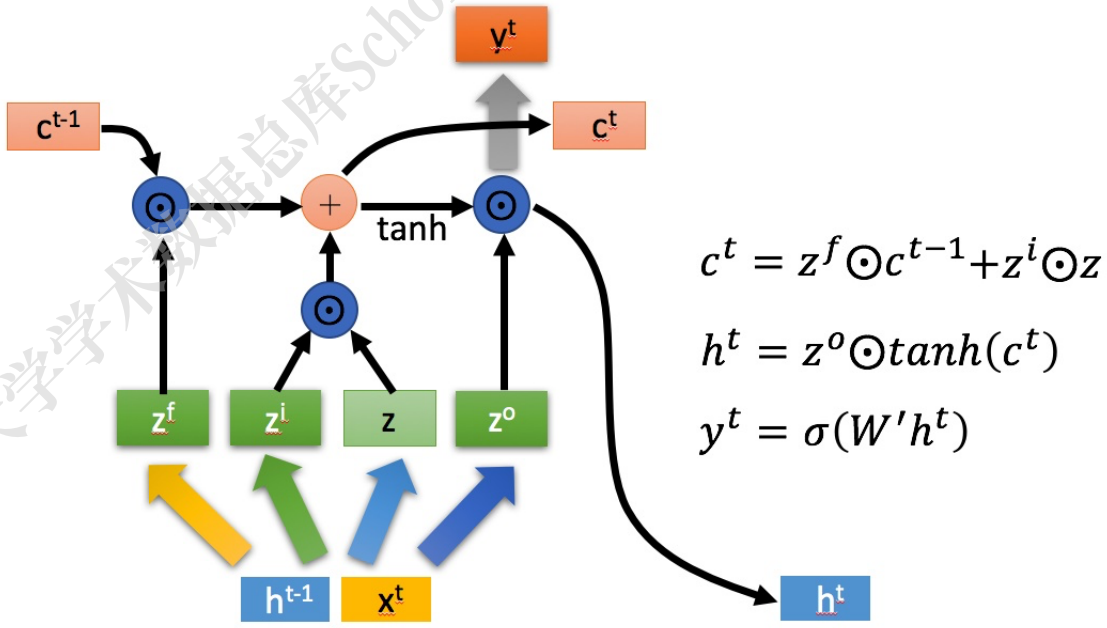

图 8：LSTM 参数图解

LSTM 具有独特的内部三个阶段：

（1）遗忘阶段

这个阶段决定了对于上一个节点传输进来的重要数据做选择性遗忘，简而言之就是记住重要的而遗忘不重要的。这个过程由内部结构中的 Forget（遗忘门）来控制上一个状态中哪些需要遗忘。

（2）选择记忆阶段

这个阶段会对输入数据进行选择性记忆，只记忆重要的信息，主要是通过内部结构中的 Information（记忆门）来控制。

（3）输出阶段。

将上面两个阶段获取的结果相加，从而得到传输给下一阶段的状态数据，也就是输出阶段，由输出门控制。这个阶段会决定最终要当成输出的状态，过程中还要对上一阶段的中间数据通过 tanh 激活函数进行放缩

4.2 股票行情数据采集

4.2.1 行情获取

本实例中选取了 2020 年 1 月 1 日到 2020 年 12 月 31 日 A 股沪深 300 指数以及成分股前复权的日级别行情 k 线数据作为实验数据。采用开源数据工具包 akshare 从互联网上下载，主要包括每个证券在对应交易日的开盘价、收盘价、最高价、最低价以及成交量等基本要素。

研究所用的数据使用沪深 300 指数主要基于的考虑是，沪深 300 指数及成分股被很多公募基金和资管所采用，包含的股票基本代表了沪深两市基本的行情走势，对于深度学习来说，在保证数据多样性的同时又能兼顾数据的全面性。

4.2.2 结构化预处理

由于从互联网直接获取的行情数据并不能直接用于模型训练分析，还需要经过结构化标准化以及异常值清洗。

首先，将获取的行情使用 python 解析后缓存到 pandas dataframe 格式中，按照标的代码、日期等对其各字段，最终存储到 csv 文件。

表 3：股票部分日行情

trade day	instname	security	open	low	high	close	volume
20200220	美的集团	000333.SZ	52.27	51.90	53.52	53.40	37041152
20200221	美的集团	000333.SZ	53.05	52.58	53.36	52.84	29779869
20200224	美的集团	000333.SZ	53.89	53.00	53.90	53.45	40782414
20200225	美的集团	000333.SZ	52.57	52.37	53.01	52.98	31546203
20200226	美的集团	000333.SZ	52.27	51.80	53.85	52.86	38724015
20200227	美的集团	000333.SZ	52.86	52.77	53.73	53.45	28493463
20200228	美的集团	000333.SZ	52.45	51.39	52.74	51.64	33405713
20200302	美的集团	000333.SZ	51.64	51.64	53.30	52.90	31893449
20200303	美的集团	000333.SZ	53.06	52.54	53.54	53.05	26499919
20200304	美的集团	000333.SZ	52.57	52.19	52.99	52.81	23908427
20200305	美的集团	000333.SZ	53.10	52.91	55.29	55.04	57571838

然后，对于部分停盘或者退市的股票行情的缺失值进行剔除或者补足处理，部分股票的交易日数据进行离散差值处理。最终保证清洗完的数据满足模型训练要求。

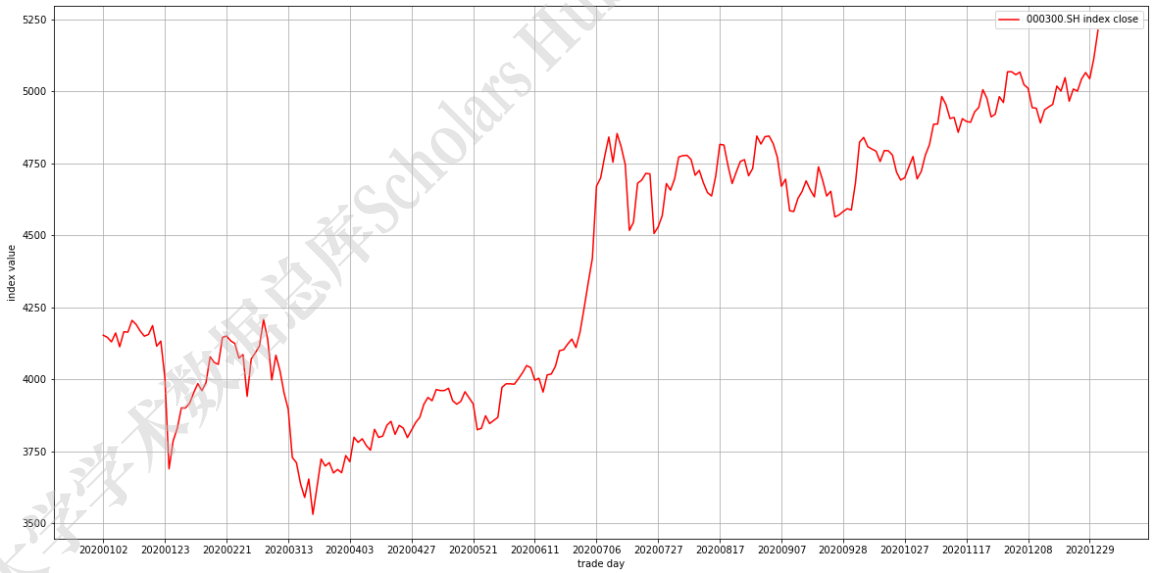

图 9：实验数据中沪深 300 指数日级别行情收盘价走势

4.3 构建 LSTM 模型量化因子

本部分使用有监督学习的深度学习方式构建股票行情走势预测模型。先将原始股票行情数据进行预处理，包括清洗插值、特征选取，然后将数据分位训练集和测

试集，基于 LSTM 构造的完整时间序列处理模型训练得到最终的股票走势预测器。最后，将训练好的模型应用于量化交易策略的涨跌预测因子。

4.3.1 数据清洗

从网络上获取的原始数据一般是不干净的，需要先经过清洗才能输入到机器学习模型进行训练，避免对于模型效果造成干扰。本章在得到最终用于深度学习数据输入所做的清洗过程主要包括下列操作：

（1）数据剔除

沪深 300 成分股，有部分股票在年中出现停牌退市或者 ST，将这部分股票剔除。

（2）数据格式标准化

日期格式统一处理成 YYMMDD 的格式，价格值统一保留两位有效数字，与金融行情终端保持对应。

（3）缺失值处理

由于数据来源于互联网，数据质量不能完全规避缺失问题，部分股票有少量交易日缺失行情价格值，需要采用插值法进行修补，为了保证数据尽量完整，并且不影响深度学习训练，本章采用均值不全方法进行插值。

4.3.2 特征选取

对于每个单只股票单日的行情数据样本，需要选择不同属性组合成一维向量作为一组特征值输入。由于不同股票的价格值不在同一个数量级，需要先将数据进行标准化，但有不能引入未来数据，所以本章考虑将每个原始数据样本的开盘价、最低价、最高价、收盘价、成交量这 5 个值的日增长复利率作为标准化的特征值，计算方式如下：

$$R_{t+1} = P_{t+1}/P_t$$

这个过程直接通过 python 的 pandas 工具包在 dataframe 结构上进行转换，代码片段如下：

```
df_feature_daily = df_stock_daily[["open_return", "low_return",
"high_return", "close_return", "volume_diff"]]
df_feature_daily = df_feature_daily.reset_index(drop=True)
```

```
df_feature_daily = df_feature_daily.drop(index=0) # drop first day
df_feature_daily
```

本章的思路是根据前 N 日的行情各价格的相对收益率预测 N+1 日行情收盘价相对收益率。那么一组数据输入就是尺寸纬度为 N x 5 的二维数组，输出就是尺寸 1 x 1 的标签。将所有的行情数据都按照这种方法处理后，就得到了带有标签的全部训练和测试用的数据集。比如，以下就是 000333.SZ 这只股票其中一个数据输入组样例：

```
[[1.0211388  0.99606234 1.0011688  0.9750628  0.7799613 ]
Input:
[0.96494156 0.9625301  0.9706471  0.9818057  1.0680736 ]
 [0.9930796  1.0214286  1.0001718  1.0148602  0.78101695]
 [1.0019164  1.002972   1.0132279  1.0039622  1.1240556 ]
 [1.0224309  1.0196967  1.0127162  1.0200754  1.1703858 ]]

Output:
1.0090833
```

4.3.3 模型构建

做完前面的准备工作之后，同样需要构建完整的深度学习模型来训练。采用 pytorch 机器学习框架的 LSTM 层叠加上线性层形成深度学习模型，模型构建主要分为两个阶段：模型训练、模型预测。

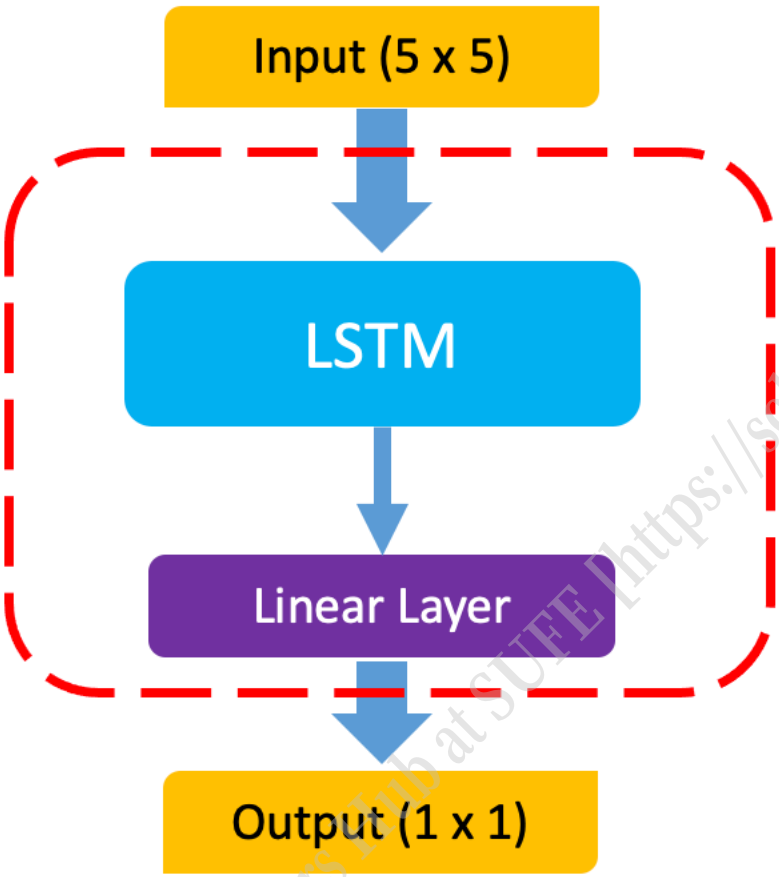

图 10：LSTM 模型结构

（1）模型训练阶段

沪深 300 成分股有若干股票，对于每只个股单独划分数据集和训练模型，将预处理好的数据划分为训练集和测试集，导入到 python 编写的时间序列预测深度学习模型进行训练，用前 5 日的数据作为每组输入来预测第 6 日的收益率，训练直到误差值收敛和准确度攀升到合理范围内结束，最后导出训练好的模型，其中每只个股的模型都是单独训练并导出的。

（2）模型预测

使用训练好的模型，将完整的所有数据集导入，输入到模型中得出输出值进行预测。这样就能得到每只股票前五日收益率对于第六日收益率的预测情况。将每批预测的值与真实值进行对比，可以得出预测的准确度。以 000333.SZ 这只股票为例，从 2020 年 1 月 1 日到 2020 年 12 月 31 日一共 242 个交易日，预处理得到的数据集

共 237 组输入和输出，当训练模型迭代 5 次之后预测准确度达到 49%，迭代 10 次之后则达到 58%，误差收敛到 0.004。

4.3.4 因子设计

深度学习 LSTM 模型分析金融行情数据并进行预测最终也是为量化投资策略设计量化因子服务的。此处，将每只股票每个交易日及前 5 日预测出来下一日的复合收益率直接作为量化因子，表示如下为：

$$NLP\_SIGNAL = R_{predict}$$

根据以上计算出来的收益率涨跌因子值，可以作为后续量化投资策略的交易信号组成元素，并参与进一步分析。

第五章 结合 NLP 和 LSTM 的量化投资多因子策略实证分析

在前面的 章节中，分别研究了基于 NLP 金融舆情文本处理的深度学习量化因子研究和基于 LSTM 金融行情时间序列的深度学习量化因子研究，为后续的整体量化投资策略提供了因子来源。本章主要阐述如何将 NLP 和 LSTM 这两种深度学习算法训练出来的因子作为信号，设计整体的多因子选股择时量化投资策略，并根据历史数据进行策略回测和评价。

5.1 策略设计

5.1.1 因子信号

根据前述的深度学习量化因子建模研究，本策略拟采用多元化因子生成股票调仓交易信号，包括：网络舆情因子、涨跌预测因子、均值回归因子。

网络舆情因子指的是，根据依据前一日股票对应的网络评论舆情文本，根据 NLP 模型推断出来的对于该股票的情绪反映，分为看好和不看好，分别量化为对应的概率值。

涨跌预测因子指的是，根据过去若干日股票对应的日级别收盘价，根据 LSTM 模型预测出来针对该股票的涨跌预测情况，看涨还是看跌，也分别量化为概率值。

均值回归因子，则是一种传统的股票因子，是在价格震荡中博取反弹的交易思路，我们认为一支股票的价格低于其均线越多的时候，它回归的可能性就越大。因此，我们可以以一支股票的价格与其均线的偏离程度作为评估标准。

由以上因子在每个交易日的实际表现组合出实际的交易信号，三种不同的因子，按照不同比例配比，进行加权汇总算出最后的实际加减仓信号。

$$S = p_1 S_1 + p_2 S_2 + p_3 S_3$$

因子信号对于交易的指示规则为：当 SIGNAL 大于指定阈值时，加仓；否则，就清空组合持仓。其中调仓的价格用当日股票的收盘价，加仓数量根据可用资金计算得出，减仓则是直接将股票当前头寸清零。

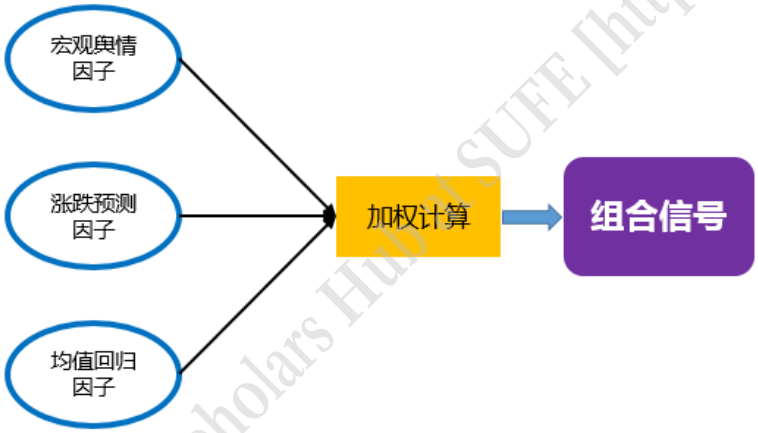

图 11：因子合成

5.1.2 交易框架流程

在因子和信号确定的情况下，需要设计整体的交易策略框架流程。

整体的交易思路如下：

（1）设定股票池，调仓频率 F，持仓股票数量 N，止盈止损线，确定因子权重参数，以及移动平均线历史交易日数量。

（2）针对每个交易日，计算出股票池所有股票的网络舆情因子、涨跌预测因子、均值回归因子，根据三种因子计算出组合信号。

（3）获取当前持仓股票，若股票的组合信号指示为减仓，则将该股票仓位清空。

（4）从股票池中取出组合信号为加仓的股票，将各股票的组合信号值进行高到低排序，取前 N 只股票进行加仓，加仓资金按照等比例配置。

在交易过程中，严格根据止盈止损线进行止盈和止损操作。

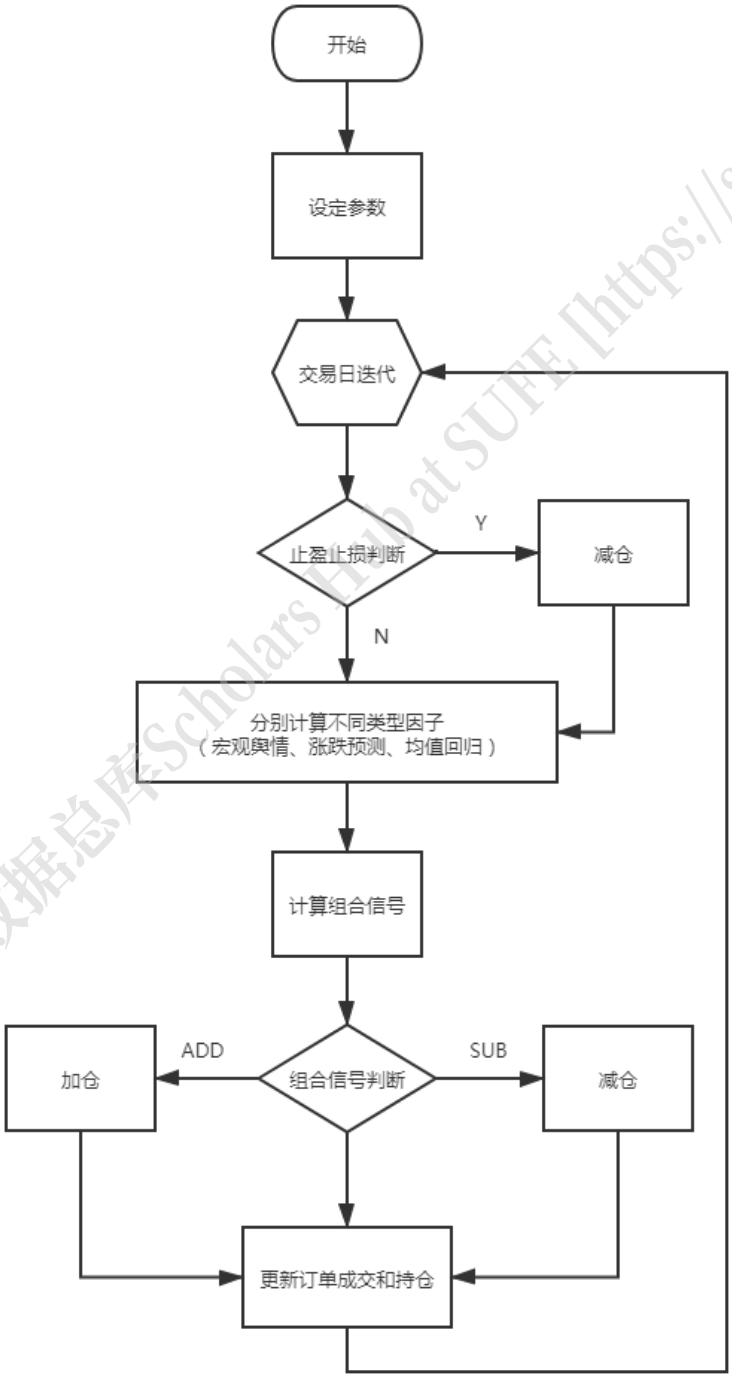

图 12：策略交易框架和流程

5.2 风险分析

策略交易过程中的证券持仓组合的组合风险可以根据 VAR（Value at Risk）风险测度进行计量。

VAR 是 JP 摩根针对其投资银行业务风险管理开发的风险测度计算方法，能够以科学实用的方式做资产配置和信息披露，在风险管理领域应用比较广泛。VAR 的完整定义是：在给定的置信度和时间间隔下，由正常市场变化

引起的高于目标水平的最大损失。也就是说，在给定时间内，证券(组合)的潜在损失不大于 VaR 的概率等于置信度。

VaR 数学表达式如下：

$$\mathrm{Prob}(\Delta P > \mathrm{VaR}) = 1 - \alpha$$

ΔP 表示组合在持有期内的价值变动量。从数学上讲，VaR 就是一个分位数。

在这里，引入 VAR 风险测度，主要是监测持仓组合的不确定性风险，将策略实际损益与策略期望损益的差距测量出来，能够从更加宏观的层面评价策略的效用。

5.3 参数敏感性分析

敏感性分析是投资项目的经济评估中常用的分析不确定性的方法之一。从多个不确定性因素中逐一找出对投资项目经济效益指标有重要影响的敏感性因素，并分析、测算其对项目经济效益指标的影响程度和敏感性程度，进而判断项目承受风险的能力。若某参数的小幅度变化能导致经济效益指标的较大变化，则称此参数为敏感性因素，反之则称其为非敏感性因素。这种分析方法的缺点是每次只允许一个因素发生变化而假定其他因素不变，这与实际情况可能不符

上述策略因子合成公式中存在权重配比，该参数不是由深度学习训练而成，是需要由人工根据实际经验调整到合适的参数，来使得策略获得最好的表现，所以在此需要对因子合成权重参数的确定进行敏感性分析。针对本文中的因子合成权重需要关注的是实际策略效果变化的百分率同参数权重的的变化百分率之比。

$$E_i = \Delta R_i / \Delta p_i$$

5.4 绩效评价

评价量化投资策略好坏的绩效指标有很多，主要包括度量收益和度量风险两方面的指标：

1.度量收益的指标

（1）总收益率

$$\mathrm{R}=\frac{V_2-V_1}{V_1}$$

V2 是截止时的资金持仓总价值，V1 是初始总价值。

(2) 年化复合收益率

不同策略的跨度时间不同，单纯从总收益率看不出实际的表现，所以需要通过年化收益率来评价。

$$R_y=(1+R)^{250/(T2-T1)}$$

其中 T2 和 T1 分别截止和起始日，一年有 250 个交易日，通过集合平均的方式计算得到年化符合收益率。

2.度量风险的指标

(1) 夏普比率

夏普比例比较的是量化策略的投资组合收益率和市场无风险收益率之间的相对插值和比例。夏普比例代表投资人没承担多少风险能获得的报酬。

$$\mathrm{Sharp}=\frac{E(Rp)-Rf}{\sigma_p}$$

(2) 最大回撤

最大回撤代表在投资期限内，任意一个时间点往前推，投资收益率从最高点跌倒最低点的区间下跌比例。最大回撤主要用来表示投资组合在一定时间内的最差情况。

$$\mathrm{drawdown}=\max(\mathrm{Di}-\mathrm{Dj})\,/\mathrm{Di}$$

5.5 策略回测

基于本文的量化投资策略构建方式，选取沪深 300 成分股作为股票池，沪深 300 指数作为基准，使用过去一整年的历史日级行情数据进行策略回测，观察策略表现。

回测实验主要是通过计算机程序编写的量化策略导入历史数据和预训练的深度学习模型运行，用到计算机软件和环境情况如下：

表 4：软件工具表

环境类别	具体配置
PC	1 台
操作系统	Linux
编程语言	Python
软件工具包	Pytorch，pandas，numpy，matplotlib
数据存储	csv

在策略回测初始化时候，要做一些初始参数的设定，包括数据相关、策略算法相关和深度学习模型相关的所有参数，设置如下：

表 5：量化策略回测参数

参数类型	参数值
回测时间段	2020 年 1 月 1 日 - 2020 年 12 月 31 日
调仓频率	每周
初始资金	200 万
股票池	沪深 300 成分股
基准	沪深 300 指数
交易费率	0.001
选股数量	20
均值回归历史天数	30
股评文本对齐词数	512
涨跌预测历史天数	5

基于前文的量化策略构建方式和参数设置运行回测程序，得到回测如下，包括部分持仓变动情况、资产价值变动情况、收益率曲线和绩效评价综合指标。

表 6：部分持仓变动

Date	Security	Side	Position
20200309	002241.SZ	LONG	600
20200309	600346.SH	LONG	800

20200309	600584. SH	LONG	500
20200309	002422. SZ	LONG	500
20200309	600332. SH	LONG	300
20200309	601607. SH	LONG	700
20200309	002304. SZ	LONG	100
20200309	002600. SZ	LONG	1100
20200316	601872. SH	LONG	13500
20200316	000963. SZ	LONG	3600
20200316	600018. SH	LONG	16000
20200316	600606. SH	LONG	8500
20200316	600011. SH	LONG	10400
20200316	601989. SH	LONG	10700
20200316	002773. SZ	LONG	2100
20200316	002739. SZ	LONG	4200
20200316	600848. SH	LONG	2300
20200316	600340. SH	LONG	2700
20200316	601857. SH	LONG	14600

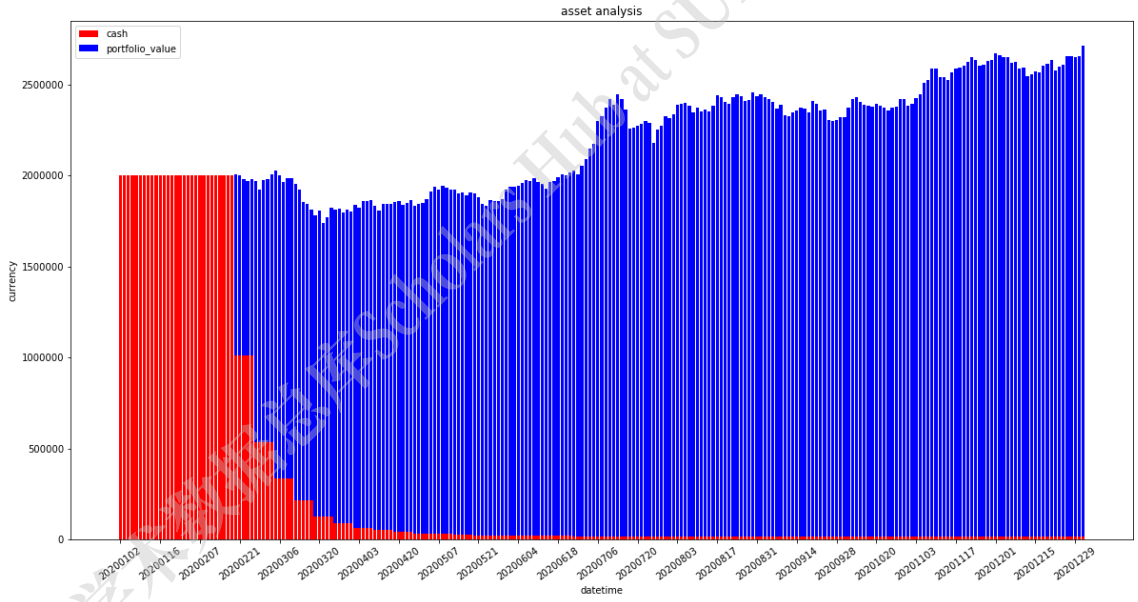

图 13：资产价值变动

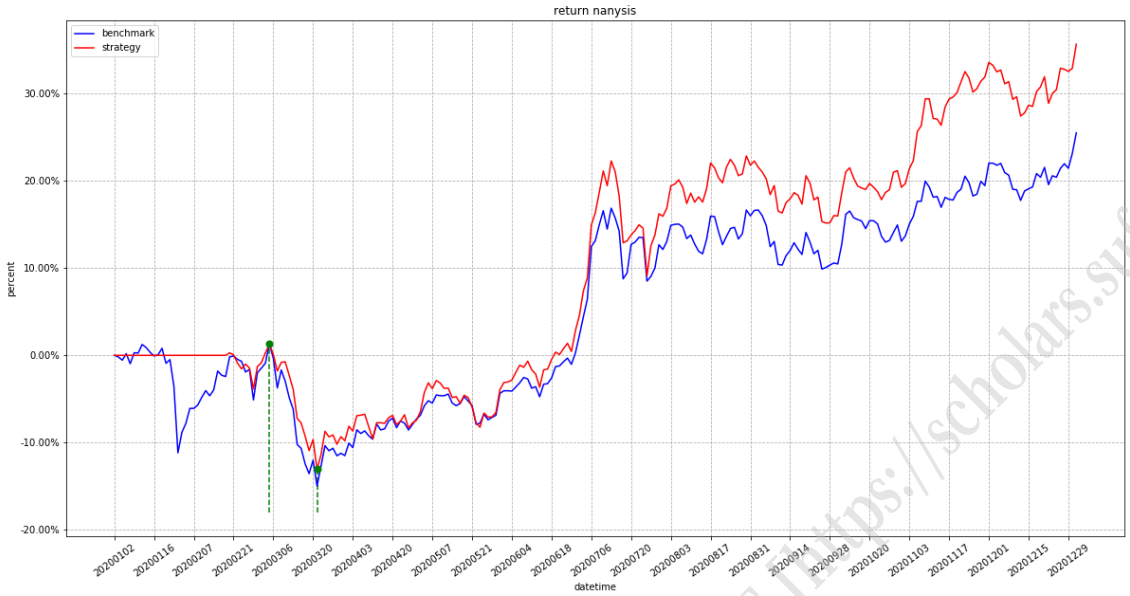

图 14：收益率和回撤曲线图

表 7：绩效评价指标

指标	数值
策略年化收益率	35.6%
基准年化收益率	25.5%
最大回撤率	14.3%
最大回测发生时间	[20200305, 20200323]
夏普比率	1.65

由以上策略回测的结果可知，结合了深度学习技术 NLP 和 LSTM 算法模型的多因子量化投资策略在 20200101 到 20201231 的沪深 300 成分股基础上进行选股和择时调仓，达到的年化总收益达到了 35.6%，高于基准收益率 25.5%，夏普比率为 1.65，在收益上的表现整体是比较高的。最大回撤达到了 14.3%，发生在沪深 300 基准下跌所在的时间段，策略回撤也小于基准本身的回撤率，风险总体可控。分析策略回测过程中的日志，发现在 2020 年疫情严重的那几个月份，网络舆情出现较多的负面情绪，能够被舆情因子捕捉到，但是由于 A 股市场的不成熟，涨跌预测因子和均值回顾因子的有效性在这段期间存在一定问题，导致没有能够中和掉网络舆情带来的下跌态势，综合计算出来的信号出现一定偏差。

在同等策略参数配置的情况下，将单因子策略与本课题的新型多因子策略进行综合比较，回测的数据结果如下。

表 8：不同策略回测结果对比

指标	均值回归策略	SVM 单因子策略	NLP 单因子策略	LSTM 单因子策略	均值回归、NLP、LSTM 多因子策略
策略年化收益率	27.3%	10.3%	12.4%	38.9%	35.6%
最大回撤率	7.9%	13.1%	12.8%	16.5%	14.3%
夏普比率	1.97	1.54	1.12	1.37	1.65

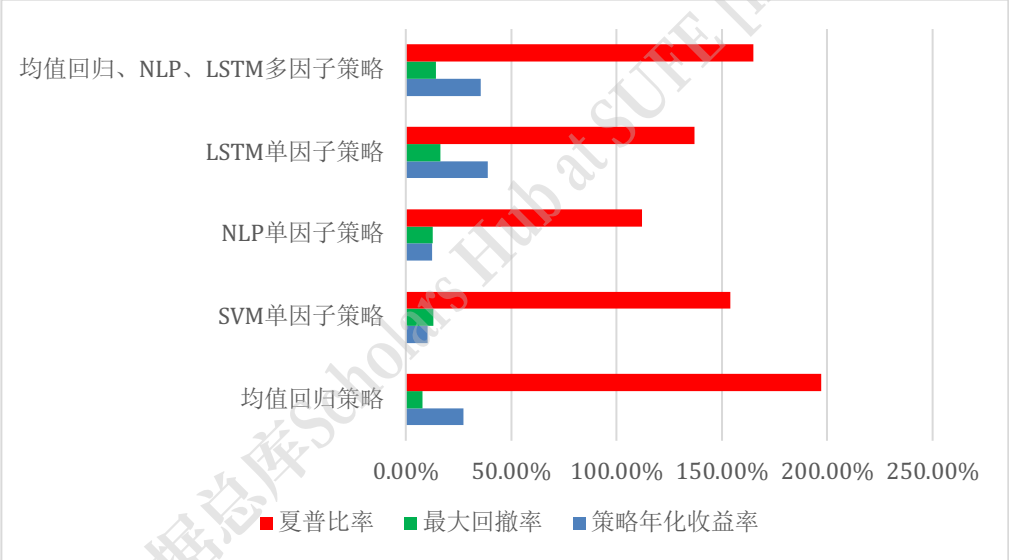

图 15：多种策略对比总览

由以上对比的数据结果可知，传统的均值回归因子和 NLP 因子策略回撤较小，收益适中，夏普比例较高；LSTM 策略则收益较高，但回撤也较高，夏普比例中等；与 NLP 和 LSTM 深度学习算法相比，传统的机器学习算法 SVM 模型构建的策略表现最不理想，收益偏低而回撤偏高。能够看出，多因子合成的策略既能取得较高的收益率，也能维持适中的回撤和夏普比例，整体的表现偏向平稳。

总体而言，该种多因子合成的策略能够在中长期市场中表现出较好的持续盈利能力，在因子信号计算方面还存在一定的改进空间。

第六章 总结与展望

6.1 总结

金融量化投资近年来成为了金融领域热门的分支，越来越多的投资者利用数学、计算机、统计等技术，充分挖掘投资资产的历史数字规律，利用这些数学规律构建量化投资策略。根据当前的现状和趋势，本文提出了一种既考虑到市场金融舆情因素也考虑到金融市场历史数据的机器学习量化交易策略。这种策略结合了自然语言处理 NLP 和时间序列处理 LSTM 两种不同的深度学习模型，提取多维度量化因子，基于深度学习算法的自然语言处理和时间序列分析在量化投资策略中的实际应用，将策略用在国内市场股票历史数据进行了实证分析。

从最终的回测结果可以看出，结合了网络舆情因子和涨跌预测因子而成的量化策略在收益率上的表现较好，风险总体可控。跟传统的量化策略相比，这种基于深度学习的量化投资策略可以挖掘出人工很难总结出来的因子和信号，从而提高策略的有效性。

6.2 展望

在研究的过程中，也遇到了一些困难，尤其是深度学习模型中神经网络参数的调整和训练的收敛，需要从各个角度不断优化。另外，深度学习应用于金融价格预测方面缺乏一些金融可解释性，在理论方面还是需要找到更加强有力的支撑。从研究的过程来看，还存在一些不足和对于未来的展望。

（1）股票因子的选择方法研究

本文选择股票因子的方法主要是采用了网络舆情和涨跌预测以及均值回归，并最终合成信号。但由于网络舆情存在样本数不足导致可能存在主观因素导致因子失真，而涨跌预测因子的数据来源容易引入未来数据。从实际应用的角度来说，可以加入更多的股票因子来综合，避免导致因子过于片面是的策略的方向性问题。

（2）股票市场交易真实性模拟研究

在做回测分析过程中，虽然考虑到了交易费率，但是回测交易过程与实际的交易还是存在一些偏差，包括滑点、行情盘口、涨跌停以及成交价等方面没有做到完

全真实模拟，回测过程比价理想化。为了保证策略的正确性，需要尽可能真实的贴近交易市场执行情况，深化更多交易细节，使得分析结果具有更好的可信度。

（3）修正损失函数的应用扩展研究

在训练深度学习模型中，用到的损失函数都是标准库函数，没有做过多的定制，不一定是训练效果最好的损失函数。网络舆情分析本质上是分类问题，而涨跌预测则是定量分析问题，针对分类问题采用的损失函数用在预测问题上则是需要修正的。

（4）模型结构的调整研究

本文进行 NLP 研究的模型是 BERT，做时间序列分析的模型是 LSTM。其中，BERT 和 LSTM 都是神经网络结构很复杂的模型，网络层级和参数调整在本文没有进行过多改动，考虑到金融策略的特殊性，可以适当地对 BERT 和 LSTM 模型进行定制，使得最终训练后效果比预期更优。

当前，量化投资和机器学习都是世界上前沿和热门的学科领域，智能投资也会逐渐占有传统投资的市场份额。深度学习可以在金融领域发挥越来越重要的作用，基于深度学习的量化投资模型算法在未来必会成为领域里值得投入的研究方向。

参考文献

[1] 高鸿桢，2001：《投资决策——技术分析与金融工程》

[2] 张弘林，2006：《西方股票投资思想的演变与当代中国股市研究》

[3] 杨健，2002：《股票市场技术分析手册》

[4] 鲍志强，1991：《证券投资技巧与理论》

[6] 潘凡，2011：《基于有效多因子的多因子选股模型》

[7] 王俊杰，2013：《量化交易在中国股市的应用》

[8] 李云飞，2008：《基于人工智能方法的股票价值投资研究》

[9] 吴荻，2011：《中国股票市场选股模型实证分析》

[10] 刘毅，2012：《因子选股模型在中国市场的实证研究》

[11] 邹运，2012：《基于遗传算法的风格选股模型研究》

[12] 江方敏，2008：《基于多因子量化模型的 A 股投资组合选股分析》

[13] 傅航聪, 张伟. 机器学习算法在股票走势预测中的应用[J]. 软件导刊, 2017,16(10): 31-34.

[14] 胡聪丛, 胡桓. 深度神经网络的发展现状[J]. 电子技术与软件工程, 2017,(4): 29-31.

[15] 刘泽光.网络舆情分析关键技术研究[D].沈阳:东北大学,2013:7-17.

[16] Carhart M M,1997,52(1):57-82,On persistence in mutual fond performance. The Journal of Finance,52(1),57-82[J]. Journal of Finance

[17] Cho K,2014,Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]

[18] Cho K,2014,Merrienboer B V, Bahdanau D, et al. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches[J]. Computer Science

[19] Chung J,2014,Gulcehre C, Cho K H, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling[J]

[20] Coates A, 2011,15:215-223,Ng A Y, Lee H. An Analysis of Single-Layer Networks in Unsupervised Feature Learning[J]. Journal of Machine Learning Research

[21] Greff K, 2016, PP(99):1-11,Srivastava R K, Koutnik J, et al. LSTM: A Search Space Odyssey[J]. IEEE Transactions on Neural Networks & Learning Systems,

[22] Gbenga Ibikunle, 2017,How random are intraday stock prices : Evidence from deep learning. Ibikunle, Gbenga; Möws, Benjamin.. p. 1-59.(Working paper)

[23] Graves A, 2013,Supervised Sequence Labelling with Recurrent Neural Networks[J].Studies in Computational Intelligence, , 385.

[24] Rosenblatt F, 1988,The perceptron: a probabilistic model for information storage and organization in the brain[M]. MIT Press,386-408.

[25] Hinton G E, 2014,Osindero S, Teh Y W. A Fast Learning Algorithm for Deep Belief Nets[J]. Neural Computation, 18(7),1527-1554.

[26] Ackert L, 2010,Deaves R.Behavioral Finance:Psychology,Decision-Making,and Markets[M].Cengage Learning,4-20.

[27] Rozeff,Michael S,1984,Dividend yields are equity risk premiums[J].The Journal of Portfolio Management,11(1):68-75.

[28] Fama E F,French K R,1988,Dividend yield and expected stock returns[J]. Journal of Financial Economics,22:3-25.

[29] Campbell J,Shiller R, 1988,The dividend-price ratio and expectations of future dividends and discount factors[J].Review of Financial Studies,1:195-228.

[30] Cochrane J, 1991,Explaining the variance of price-dividend ratios[J].Review of Financial Studies, 5(2):243-280.

[31] Hodrick R, 1992,Dividend yields and expected returns:Alternative procedures for inference and measurement[J].Review of Financial Studies,5:357-386.

[32] Lewellen J, 2004,Predicting returns with financial ratios[J].Journal of Financial Economics, 74(2):209-235.

致谢

时间过的很快，岁月如梭，短短三年，我的在职硕士生涯快到终章。在这毕业之际，心中充满了对教导我的老师们和同学们的感恩，也充满了很多的依依不舍。在此，我要像这不平凡的在职研究生经历中给我帮助的老师和同学们致以诚挚的感谢。

首先，感谢我的论文指导老师曹老师，感谢曹老师在我论文开题和论文撰稿中给予我的悉心指导和鼓励。曹老师的专业学术态度和渊博的知识留给我深刻的印象，不仅是在量化投资领域也在人生哲理上都对我有很深的引导作用。

其次，感谢这三年来管理我们在职金融班的洪老师，她是我们的班主任，在平时的学习和考试已经各项学籍管理上不辞劳苦，默默付出，明确了我读在职研究生的目标和方向。还要感谢金融班这个大集体的班长姜同学和刘同学，她们为我们的考前复习和论文写作方面提供了大量的资料辅导和指引，让我们面对众多学习和考试任务时不再迷茫，也要感谢金融班里一起学习考试的同学们，这是有了他们，学习生涯才不再孤独，可以深入的交流和沟通。

最后，我要感谢这些年来陪伴我的伴侣和我的父母，她们是我求学背后坚强的后盾，在生活上为分担了许多，辛苦奉献了许多，她们的鼓励和支持也是我坚持完成学业的支柱。

最后，感谢参与我论文答辩评审的老师。

附录

附录 A：代码

深度学习训练主要代码

```
# construct comment features and labels
origin_comments = []
origin_close_returns = []

for index, row in df_stock_comments.iterrows():
    date = row["date"]
    comment = row["comment"]

    if date in next_day_close_return_map:
        # make sure comments and returns have the same length
        origin_comments.append(comment)
        origin_close_returns.append(next_day_close_return_map[date])

print ("origin_comments:", origin_comments)
print ("origin_close_returns:", origin_close_returns)

# tokenize
def pad(x):
    # pad every text ids to the same length
    max_len = 512
    return x[:max_len] if len(x) > max_len else x + [0] * (max_len -
len(x))

def get_one_hot(label, size=2):
    # convert 1 or 0 to one hot scalar
    vec = [0] * size
    vec[label] = 1
    return vec

features = [tokenizer.encode(text) for text in origin_comments]
features = [pad(x) for x in features]

labels = [1 if r >= 1 else 0 for r in origin_close_returns]
labels = [get_one_hot(label) for label in labels]

# train data and labels dataset
```

```
train_dataset_ratio = 0.7
batch_size = 4

total_len = len(features)
train_features = features[:int(train_dataset_ratio * total_len)]
train_labels = labels[:int(train_dataset_ratio * total_len)]
test_features = features[int(train_dataset_ratio * total_len):]
test_labels = labels[int(train_dataset_ratio * total_len):]

# here input use integer, label use float
train_dataset = Data.TensorDataset(torch.LongTensor(train_features),
torch.FloatTensor(train_labels))
test_dataset = Data.TensorDataset(torch.LongTensor(test_features),
torch.FloatTensor(test_labels))

train_iter = Data.DataLoader(train_dataset, batch_size, shuffle=True)
test_iter = Data.DataLoader(test_dataset, batch_size, shuffle=True)
# define DL model
class TextBert(nn.Module):
    def __init__(self):
        super(TextBert, self).__init__()
        self.model = bert_model
        self.dropout = nn.Dropout(0.1) # to avoid overfit
        self.linear = nn.Linear(768, 2) # output size as 2

    def forward(self, x, attention_mask=None):
        y = self.model(x, attention_mask=attention_mask)
        outputs = y[1] # get result after pool, shape: batch * 768
        outputs = outputs.view(-1, 768)
        outputs = self.dropout(outputs)
        outputs = self.linear(outputs)
        return outputs

# training  (take very long time)
num_epochs = 5
for epoch in range(num_epochs):
    train_loss_sum, batch_count = 0.0, 0
    for feature, label in train_iter:
        output = model(feature)
        train_loss = loss(sigmoid(output).view(-1, 2), label.view(-1,
2))
```

```
        optimizer.zero_grad()
        train_loss.backward()
        optimizer.step()

        train_loss_sum += train_loss.item()
        batch_count += 1
    print ("epoch %d, train loss: %.4f" % (epoch + 1, train_loss_sum /
batch_count))
```

```
df_feature_daily = df_stock_daily[["open_return", "low_return",
"high_return", "close_return", "volume_diff"]]
df_feature_daily = df_feature_daily.reset_index(drop=True)
df_feature_daily = df_feature_daily.drop(index=0) # drop first day
print ("feature columns:", df_feature_daily.columns.to_list())
df_feature_daily

# prepare train and test data
step = 5 # previous days
features = []
labels = []
for i in range(len(df_feature_daily) - step):
    features.append(df_feature_daily.iloc[i:(i + step)].values)
    labels.append(df_feature_daily.iloc[(i + step)]["close_return"])

features = np.array(features, dtype=np.float32) # better to specify
dtype
labels = np.array(labels, dtype=np.float32)

train_dataset_ratio = 0.7
batch_size = 10

total_len = len(features)
train_features = features[:int(train_dataset_ratio * total_len)]
train_labels = labels[:int(train_dataset_ratio * total_len)]
test_features = features[int(train_dataset_ratio * total_len):]
test_labels = labels[int(train_dataset_ratio * total_len):]

train_dataset = Data.TensorDataset(torch.Tensor(train_features),
torch.Tensor(train_labels))
```

```
test_dataset = Data.TensorDataset(torch.Tensor(test_features),
torch.Tensor(test_labels))
train_iter = Data.DataLoader(train_dataset, batch_size, shuffle=True)
test_iter = Data.DataLoader(test_dataset, batch_size, shuffle=True)

# define DL model
class StockLSTM(nn.Module):
    def __init__(self, input_size=5, output_size=1, hidden_size=32,
num_layers=1):
        super(StockLSTM, self).__init__()

        # lstm input shape: (batch, seq_len, input_size)
        self.lstm = nn.LSTM(input_size=input_size,
                            hidden_size=hidden_size,
                            num_layers=num_layers,
                            batch_first=True)
        self.linear = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        # hn.shape and cn.shape : num_layers * direction_numbers, batch,
hidden_size
        r_out, (h_n, h_c) = self.lstm(x)
        a, b, c = h_n.shape
        out = self.linear(h_n.reshape(a * b, c)) # must do the reshape
        return out

# training
for epoch in range(num_epochs):
    train_loss_sum = 0
    batch_count = 0
    for _, (feature, label) in enumerate(train_iter):
        output = net(feature) # maybe should use feature.squeeze(1) as
input
        train_loss = loss(output, label.unsqueeze(1)) # should add dim
        optimizer.zero_grad()
        train_loss.backward()
        optimizer.step()

        train_loss_sum += train_loss.item()
        batch_count += 1
```

```
    print ("epoch %d, train loss: %.4f" % (epoch + 1, train_loss_sum / batch_count))
```

量化交易策略主要代码

```
class StrategyBacktest:
    def __init__(self, strategy_name):
        self._strategy_name = strategy_name
        self._pre_trading_day = 0
        self._current_trading_day = 0
        self._his_data_series = [] # history trading day series which already in sequence
        self._his_data_map = {} # trading_day -> {security -> bar_data}
        self._start_trading_day = 0
        self._end_trading_day = 0
        self._current_cash = 0.0
        self._trading_cost_ratio = 0.0
        self._cash_map = {} # trading_day -> cash
        self._portfolio_value_map = {} # trading_day -> porfolio value
        self._positions_map = {} # trading_day -> positions
        self._benchmark = "" # the basic index benchmark
        self._universe = [] # available trading security pool
        self._adjust_freq = 5 # adjust position frequency trading days

        # factor weight
        self._nlp_factor_weight = 0.4
        self._lstm_factor_weight = 0.3
        self._ma_factor_weight = 0.3

        # analysis result to show
        self.asset_file_path = ""
        self.return_file_path = ""
        self.criteria_file_path = ""
        self.asset_plot_path = ""
        self.return_plot_path = ""

    def init(self, history_file_path):
        # load history data
        self._load_data(history_file_path)

        # financial settings
```

```
        self._start_trading_day = 20200102 # should be real trading day
and adapt to his data series
        self._end_trading_day = 20201231
        self._current_cash = 2000000
        self._trading_cost_ratio = 0.001
        self._benchmark = "000300.SH"
        self._universe = security_universe

        # strategy specific params
        self.ma_days = 30 # moving average days
        self.stock_num = 20 # every adjust cycle choose 10 stocks

        # preprocess data, NLP, LSTM, moving average value
        self.ma_dict = dict()
        self.nlp_dict = dict()
        self.lstm_dict = dict()
        for security in self._universe:
            df_security = df_his_data[df_his_data["security"] ==
security].copy() # should use copy

            nlp_features_tensor =
torch.LongTensor(nlp_stock_comments_features)
            df_security["nlp"] = nlp_net(nlp_features_tensor)

            lstm_features_tensor =
torch.Tensor(lstm_stock_marketdata_features)
            df_security["lstm"] = lstm_net(lstm_features_tensor)

            df_security["ma30"] = ta.MA(df_security["close_price"],
self.ma_days)
            self.ma_dict[security] = df_security
        print ("--- preprocess data success, total ma_dict size:",
len(self.ma_dict))

        # NLP and LSTM model preprocess

    def _load_data(self, history_file_path):
        # construct his data structure, preparing for later usage
        print ("--- loading history data")
        df_his_data = pd.read_csv(history_file_path)
```

```
        date_series = df_his_data["trading_day"].values.tolist()
        date_series = list(set(date_series))
        date_series.sort()
        self._his_data_series = date_series

        for index, row in df_his_data.iterrows():
            trading_day = row["trading_day"]

            if trading_day not in self._his_data_map:
                self._his_data_map[trading_day] = {}

            self._his_data_map[trading_day][row["security"]] = {
                "trading_day": trading_day,
                "security": row["security"],
                "open_price": row["open_price"],
                "low_price": row["low_price"],
                "high_price": row["high_price"],
                "close_price": row["close_price"],
                "volume": row["volume"]
            }
        # print (self._his_data_map)
        print ("--- history data loaded success, total trading days:",
len(self._his_data_series))

    def _handle_market_data(self, market_data_group):

        # ---- write strategy code here ---- #
#         for security in market_data_group:
#             print (security, market_data_group[security])

        # rank the (ma - close) / close and choose the top 10 stocks to
buy and sell all other not inside
        trading_day = 0
        factor_array = []
        for security, market_data in market_data_group.items():
            if security == self._benchmark:
                continue

            trading_day = market_data["trading_day"] # here assign
trading_day
            close_price = market_data["close_price"]
```

```
            df_security = self.ma_dict[security]
            df_ma = df_security[df_security["trading_day"] ==
trading_day]
            if len(df_ma) > 0:
                ma = df_ma.iloc[0]["ma30"]
                if not np.isnan(ma):
                    # nlp factor
                    mood_value = df_ma.iloc[0]["nlp"]
                    # lstm factor
                    predict_ratio = df_ma.iloc[0]["lstm"]
                    # moving averate factor
                    price_diff = (ma - close_price) / close_price

                    # combine to final factor
                    combine_factor = self._nlp_factor_weight *
mood_value + self._lstm_factor_weight * predict_ratio +
self._ma_factor_weight * price_diff

                    factor_array.append({
                        "security": security,
                        "combine_factor": combine_factor
                    })
        factor_array = sorted(factor_array, key=(lambda x: -
x["combine_factor"])) # rank by price diff

        if len(factor_array) == 0:
            print ("factor array empty, not signal, do nothing")
            return
        elif len(factor_array) > self.stock_num:
            factor_array = factor_array[:self.stock_num] # get top 10
stocks

        print ("ranked top factor array: ", factor_array)

        position_map = self._positions_map[trading_day]

        # sell security that not in factor array
        for position_key in position_map:
            security = position_key
```

```
            if security not in market_data_group:
                continue

            market_data = market_data_group[security]
            open_price = market_data["open_price"]
            close_price = market_data["close_price"]

            if security not in factor_array:
                price = close_price # use close price
                quantity = position_map[position_key]["pre_size"] / 2 #
sell some enable position

                if quantity > 0:
                    self._place_order(ACTION_SELL, security, price,
quantity)

        # buy security that in factory array
        cash_part = self._current_cash / 2 / len(factor_array) # decide
cash to do invest
        for factor_item in factor_array:
            security = factor_item["security"]
            market_data = market_data_group[security]

            open_price = market_data["open_price"]
            close_price = market_data["close_price"]

            price = close_price # use close price
            quantity = int(cash_part / price / 100) * 100 # buy max size
and consider lotsize

            if quantity > 0:
                self._place_order(ACTION_BUY, security, price, quantity)

    def _place_order(self, action, security, price, quantity):
        print ("$$ place order, action: %s price: %f quantity: %d" %
(action, price, quantity))

        # assume every order will be filled
        # simulate order, adjust position and asset
        if action == ACTION_BUY:
            if price * quantity <= self._current_cash:
```

```
                print ("update cash")
                self._current_cash -= price * quantity * (1 +
self._trading_cost_ratio) # consider trading cost
                print ("current cash is:", self._current_cash)

                print ("update position")
                security_direction = security + '_' + SIDE_LONG
                if security_direction not in
self._positions_map[self._current_trading_day]:
                    # new position
                    position = {
                        "trading_day": self._current_trading_day,
                        "security": security,
                        "direction": SIDE_LONG,
                        "pre_size": 0,
                        "current_size": quantity
                    }

self._positions_map[self._current_trading_day][security_direction] =
position
                else:

self._positions_map[self._current_trading_day][security_direction]["curr
ent_size"] += quantity
            else:
                print ("current cash %f not enough, order price: %f
quantity: %d" % (self._current_cash, price, quantity))
        elif action == ACTION_SELL:
            security_direction = security + '_' + SIDE_LONG
            if security_direction in
self._positions_map[self._current_trading_day]:
                position =
self._positions_map[self._current_trading_day][security_direction]
                if position["pre_size"] >= quantity:
                    print ("update cash")
                    self._current_cash += price * quantity * (1 -
self._trading_cost_ratio) # consider trading cost
                    print ("current cash is:", self._current_cash)

                    print ("update position")
```

```
self._positions_map[self._current_trading_day][security_direction]["pre_
size"] -= quantity

self._positions_map[self._current_trading_day][security_direction]["curr
ent_size"] -= quantity

                    # if position empty, remove it
                    if
self._positions_map[self._current_trading_day][security_direction]["curr
ent_size"] == 0:
                        del
self._positions_map[self._current_trading_day][security_direction]
                else:
                    print ("no enough position to sell, security: %s
pre_size: %d" % (security, position["pre_size"]))
            else:
                print ("position not exist, security: %s" % security)

    def _roll_position(self):
        if self._pre_trading_day == 0:
            print ("the first trading day, skip rolling")
            return

        print ("roll positions to current trading day")

        # init current trading day position with pre trading day
position
#         self._positions_map[self._current_trading_day] =
self._positions_map[self._pre_trading_day] # FIXME: this is wrong
        self._positions_map[self._current_trading_day] = {} # FIXME:
have to copy the item one by one to avoid reference error
        for security_position_key in
self._positions_map[self._pre_trading_day]:

self._positions_map[self._current_trading_day][security_position_key] =
{}

self._positions_map[self._current_trading_day][security_position_key]["t
rading_day"] = self._current_trading_day
```

```
self._positions_map[self._current_trading_day][security_position_key]["s
ecurity"] =
self._positions_map[self._pre_trading_day][security_position_key]["secur
ity"]

self._positions_map[self._current_trading_day][security_position_key]["p
re_size"] =
self._positions_map[self._pre_trading_day][security_position_key]["curre
nt_size"]

self._positions_map[self._current_trading_day][security_position_key]["c
urrent_size"] =
self._positions_map[self._pre_trading_day][security_position_key]["curre
nt_size"]

        print ("roll asset to current trading day")

        self._cash_map[self._current_trading_day] =
self._cash_map[self._pre_trading_day]
        self._portfolio_value_map[self._current_trading_day] =
self._portfolio_value_map[self._pre_trading_day]

    def _update_asset(self, market_data_group):
        # update asset map
        portfolio_value = 0.0 # mark to close price
        for position in
self._positions_map[self._current_trading_day].values():
            if position != {} and position["security"] in
market_data_group:
                # make sure the security has market data at this day
                portfolio_value += position["current_size"] *
market_data_group[position["security"]]["close_price"]

        self._cash_map[self._current_trading_day] = self._current_cash
        self._portfolio_value_map[self._current_trading_day] =
portfolio_value

    def _analyze(self):
        print ("==== do analyze ====")
```

```
        print( "--- calculate analysis result")
        # return analysis
        benchmark_returns_map = {} # date in sequence
        strategy_returns_map = {} # date in sequence
        benchmark_total_return = 0.0
        strategy_total_return = 0.0

        first_trading_day = list(self._cash_map.keys())[0]
        last_trading_day = list(self._cash_map.keys())[-1]
        for trading_day in self._cash_map.keys():
            if trading_day == first_trading_day:
                benchmark_returns_map[trading_day] = 0.0
                strategy_returns_map[trading_day] = 0.0
            else:
                benchmark_init_price =
self._his_data_map[first_trading_day][self._benchmark]["close_price"]
                benchmark_current_price =
self._his_data_map[trading_day][self._benchmark]["close_price"]
                benchmark_return = (benchmark_current_price -
benchmark_init_price) / benchmark_init_price
                benchmark_returns_map[trading_day] = benchmark_return

                strategy_init_asset = self._cash_map[first_trading_day]
+ self._portfolio_value_map[first_trading_day]
                strategy_current_asset = self._cash_map[trading_day] +
self._portfolio_value_map[trading_day]
                strategy_return = (strategy_current_asset -
strategy_init_asset) / strategy_init_asset
                strategy_returns_map[trading_day] = strategy_return

        benchmark_total_return = benchmark_returns_map[last_trading_day]
        strategy_total_return = strategy_returns_map[last_trading_day]

        # other criteria
        # TODO: alpha, beta, sharp

        # max drawdown
        max_drawdown = 0.0
        max_drawdown_begin_day = 0
        max_drawdown_end_day = 0
```

```
        pre_max_return = 0.0
        pre_max_day = 0

        for (trading_day, strategy_return) in
strategy_returns_map.items():
            drawdown = pre_max_return - strategy_return
            if drawdown > max_drawdown:
                max_drawdown_begin_day = pre_max_day
                max_drawdown_end_day = trading_day
                max_drawdown = drawdown

            if strategy_return > pre_max_return:
                pre_max_day = trading_day
                pre_max_return = strategy_return

        max_drawdown_group = {
            "max_drawdown": max_drawdown,
            "max_begin_day": max_drawdown_begin_day,
            "max_end_day": max_drawdown_end_day
        }

        print ("write analysis file")
        timestamp = datetime.now().strftime('%Y-%m-%d_%H_%M_%S')
        result_dir = "result_" + timestamp
        os.mkdir(result_dir)

        # position
        position_file_path = result_dir + '/' + "position_" +
self._strategy_name + '_' + timestamp + ".csv"
        with open(position_file_path, 'w') as f:

f.write("trading_day,security,direction,pre_size,current_size\n") #
title line
            for (trading_day, positions) in self._positions_map.items():
                for (security_position_key, position) in
positions.items():
                    line = str(position["trading_day"]) + ',' +
position["security"] + ',' + SIDE_LONG + ',' + str(position["pre_size"])
+ ',' + str(position["current_size"]) + "\n"
                    f.write(line)
        print ("saved position file to path:", position_file_path)
```

```
        # asset
        self.asset_file_path = result_dir + '/' + "asset_" +
self._strategy_name + '_' + timestamp + ".csv"
        with open(self.asset_file_path, 'w') as f:
            f.write("trading_day,cash,portfolio_value,total\n")
            for trading_day in self._cash_map.keys():
                cash = self._cash_map[trading_day]
                portfolio_value = self._portfolio_value_map[trading_day]
                total = cash + portfolio_value
                line = str(trading_day) + ',' + str(cash) + ',' +
str(portfolio_value) + ',' + str(total) + "\n"
                f.write(line)
        print ("saved asset file to path:", self.asset_file_path)

        # return
        self.return_file_path = result_dir + '/' + "return_" +
self._strategy_name + '_' + timestamp + ".csv"
        with open(self.return_file_path, 'w') as f:
            f.write("trading_day,benchmark_return,strategy_return\n")
            for trading_day in strategy_returns_map.keys():
                line = str(trading_day) + ',' +
str(benchmark_returns_map[trading_day]) + ',' +
str(strategy_returns_map[trading_day]) + "\n"
                f.write(line)
        print ("saved return file to path:", self.return_file_path)

        # criteria
        self.criteria_file_path = result_dir + '/' + "criteria_" +
self._strategy_name + '_' + timestamp + ".csv"
        with open(self.criteria_file_path, 'w') as f:

f.write("strategy_total_return,benchmark_total_return,alpha,beta,sharp,m
ax_drawdown,max_drawdown_begin_day,max_drawdown_end_day\n")
            line = str(strategy_total_return) + ',' +
str(benchmark_total_return) + ',' + "0.0,0.0,0.0," + str(max_drawdown) +
',' + str(max_drawdown_begin_day) + ',' + str(max_drawdown_end_day)
            f.write(line)
        print ("saved criteria file to path:", self.criteria_file_path)

        print ("--- plot analysis result")
        # plot asset analysis
```

```
        max_drawdown_begin_x = str(max_drawdown_begin_day)
        max_drawdown_end_x = str(max_drawdown_end_day)
        max_drawdown_x_array = [max_drawdown_begin_x,
max_drawdown_end_x]

        max_drawdown_y_array = []
        if max_drawdown_begin_day in strategy_returns_map and
max_drawdown_end_day in strategy_returns_map:
            max_drawdown_begin_y =
strategy_returns_map[max_drawdown_begin_day]
            max_drawdown_y_array.append(max_drawdown_begin_y)
            max_drawdown_end_y =
strategy_returns_map[max_drawdown_end_day]
            max_drawdown_y_array.append(max_drawdown_end_y)

        self.asset_plot_path = result_dir + '/' + "asset_plot_" +
self._strategy_name + '_' + timestamp + ".png"
        trading_day_label = [str(trading_day) for trading_day in
self._cash_map.keys()] # must use string label as x axis
        plt.figure(figsize=(20, 10))
        plt.xticks(range(0, len(trading_day_label), 10), rotation=36) #
make label text less, rotate the label to optimize display
        plt.title("asset analysis")
        plt.bar(trading_day_label, self._cash_map.values(), color='r',
label="cash") # here no need to convert to list
        plt.bar(trading_day_label, self._portfolio_value_map.values(),
bottom=list(self._cash_map.values()), color='b',
label="portfolio_value")
        plt.xlabel("datetime")
        plt.ylabel("currency")
        plt.legend()
#         plt.show()
        plt.savefig(self.asset_plot_path)
        print ("saved asset plot img to path:", self.asset_plot_path)

        # plot return analysis
        plt.clf()

        self.return_plot_path = result_dir + '/' + "return_plot_" +
self._strategy_name + '_' + timestamp + ".png"
```

```
        def to_percent(value, position):
            return '%1.2f' % (value * 100) + '%'
        plt.gca().yaxis.set_major_formatter(FuncFormatter(to_percent))

        plt.xticks(range(0, len(trading_day_label), 10), rotation=36)
        plt.grid(ls='--')
        plt.title("return nanysis")
        plt.plot(trading_day_label,
list(benchmark_returns_map.values()), "b-", label="benchmark") # here
must convert to list
        plt.plot(trading_day_label, list(strategy_returns_map.values()),
"r-", label="strategy")

        # make sure dots exist
        if len(max_drawdown_y_array) == 2:
            plt.plot(max_drawdown_x_array, max_drawdown_y_array, "go",
markersize=7)
            plt.vlines(max_drawdown_begin_x, max_drawdown_end_y - 0.05,
max_drawdown_begin_y, 'g', "--")
            plt.vlines(max_drawdown_end_x, max_drawdown_end_y - 0.05,
max_drawdown_end_y, 'g', "--")

        plt.legend()
        plt.xlabel("datetime")
        plt.ylabel("percent")
#         plt.show()
        plt.savefig(self.return_plot_path)
        print ("saved return plot img to path:", self.return_plot_path)

    def run(self):
        print ("==== start running backtest ====")

        # loop his data and adjust all the staff day by day
        day_count = 1
        for current_trading_day in self._his_data_series:
            self._pre_trading_day = self._current_trading_day
            self._current_trading_day = current_trading_day
            print ("--- [%d] current trading day, pre trading day is
[%d]---" % (self._current_trading_day, self._pre_trading_day))

            # lazy load
```

```
            if self._current_trading_day not in self._positions_map:
                self._positions_map[self._current_trading_day] = {}

            if self._current_trading_day not in self._cash_map:
                self._cash_map[self._current_trading_day] =
self._current_cash # use number

            if self._current_trading_day not in
self._portfolio_value_map:
                self._portfolio_value_map[self._current_trading_day] = 0
# use number

            # before adjust position
            self._roll_position()

            bar_data_group = self._his_data_map[current_trading_day]

            if day_count % self._adjust_freq == 0:
                self._handle_market_data(bar_data_group) # multiple
securities drive at once

            # after adjust position
            self._update_asset(bar_data_group)

            # increase
            day_count += 1

        self._analyze()

        print ("==== end running backtest ====")
```