基于伊利股份的季节性时间序列预测模型

——数据结构与算法大作业

姓　　名：郑文韬、李尚文、李心珺

学　　号：1512304，1512316，1512317

学　　院：商学院

专　　业：国际会计

完成日期：2018 年 6 月 5 日

摘要

本文对伊利股份公司营业成本、营业收入、利润和股价多个因变量建立预测模型，进行了数据预处理，因子筛选与合成降维，模型搭建等操作。

首先是对数据的预处理，预处理可以分为三个阶段：1、数据清洗：利用 python 进行对原始数据异常值、缺失值、非相关值、重复值的清洗，保证数据的稳定性；2、数据对齐：考虑到所提供数据中基本均为不同发布频率的原始数据，为了保证数据的有效性和合理性，这里采用了近邻值的方法、分别以两个档次因变量数据为参考序列实现了数据的对齐。3、数据转换：考虑到数据之间数量级有相当大的差异，为有效的构建预测模型，在分别对归一化、规范化和 z 分位数标准化尝试进行数值变换的过程之后，发现 z 分位数标准化的预测效果更显著，因而采用 z 分位数标准化的方式进行数值转换。

其次是对因子的筛选与合成降维：因子筛选分别采用了筛除低波动，单变量筛选，Lasso 正则化、随机 Lasso 和贪心算法等方式进行因子筛选。

接下来便是是分别对两个档次进行预测模型的搭建：

一、考虑对营业成本、营业收入、利润建立预测模型：结合对实际数据的初步判断，发现伊利的营业成本、营业收入、利润具有很强的季节效应，拟建立具有季节效应的时间序列模型。二、考虑对伊利股价建立预测模型：结合对实际数据的初步判断，伊利的股价序列为日频，同时观察到伊利股价 5000+的样本量，考虑到机器学习在大样本量学习优化建立预测模型的优势，同时考虑到深度学习，这里拟采用长短期记忆神经网络 RNN-LST 来建立预测模型。

最后是对模型的展望，上述模型存在如下可以拓展的地方：在构建时间序列模型当中，可以考虑加入到其它一些外生变量作为因子，构造 SVARIMAX 时间序列预测模型，同时后续可以考虑运用改进的 CNN 神经网络的方式建立预测模型等。

关键词：LSTM; SVARIM; Lasso; 深度学习；时间序列；随机森林

目录

第一章 导论

1.1 研究背景

自从资本市场存在以来，对各个投资策略的投资收益的预测就成为投资者和研究者十分关心的问题。

公司的利润是投资者和公司管理层十分关心的问题。会计中的利润是指企业在一定会计期间的经营成果。利润包括收入减去费用后的净额、直接计入当期利润的利得和损失等。利润按其构成的不同层次可划分为：营业利润、利润总额和净利润。利润是衡量企业优劣的一种重要标志，往往是评价企业管理层业绩的一项重要指标，也是投资者等财务报告使用者进行决策时的重要参考。目前，预测公司利润率的方法主要有包括回归在内的统计学方法。

股票价格的预测一直是非常困难的问题。学界目前已提出了不少相关的股票定价理论。传统的定价理论着重于价值发现功能,从企业角度入手考察股票价格决定因素;现代定价理论则从投资者的角度出发,更多地考虑了投资的现实情况:投资者往往不是投资于一种股票,而是投资于多种股票而形成的投资组合。

传统股票定价理论主要指稳固基础理论,其基本思想是,股票具有其内在价值,它是股票价格的稳固基点。股票价格以股票的内在价值为基础,并决定于内在价值,当股票市价高于(或低于)其内在价值时,就出现卖出(或买进)机会,股票价格总是围绕其内在价值而上下波动的。因而稳固基础理论也称为内在价值理论。目前流行的、较有影响的估价方法主要有三种: 现金流贴现模型、相对估价法和经济附加值法

现代股票定价理论成为一个专门研究领域确立下来是许多经济学家共同努力的结果,早期的工作主要是对不确定条件下量化模型的建立提出一些构想。例如,费雪于 1932 年首次提出了未来资产收益的不确定条件下可以用概率分布来描述的观点。此后,马夏克、希克斯等学者经过一系列的研究,认为投资者的投资偏好可以看作是对投资于未来收益的概率分布的偏好,可以用均方差空间的无差异曲线来表示,并认为“大数定律”在包含多种风险资产投资中将发挥某种作用,此外,还提出了风险溢价这一重要概念。现代股票定价理论主要包括：现代证券组合理论、资本资产定价模型、因素模型和套利定价理论

现在，由于数据科学的兴起，各种机器学习方法因其处理大量复杂数据的能力被应用于股票价格的预测。虽然它们没有依赖于任何经济学理论，但在预测时也取得了较好的效果。

1.2 研究目的

本文使用伊利公司作为案例，收集了大量与伊利公司业务相关的因子数据。并使用统计和机器学习的模型对现有伊利股份公司的收益和股价进行拟合。拟合出的模型可以用于预测未来伊利股份公司的收益和股票价格。

我们的拟合模型可以解决与伊利股份公司有关的决策问题。对于金融中介而言，我们的模型旨在帮助投资公司使用此模型对伊利股份公司做出收益预测，并预测未来的股票价格变动。对于投资者而言，我们的模型旨在帮助投资者根据我们对伊利盈利能力和股票价格的分析做出更理智的决策。对于伊利股份公司本身而言，我们的模型旨在帮助公司高管进行绩效评价和根据预测值进行未来公司营业目标的制定。

不仅如此，我们希望建立的模型可以应用于其他股票和公司的投资收益的预测和分析，解决通过相关因子股票预测和公司绩效预测的问题。

第二章 统计与机器学习方法的理论综述

本章为导论部分，主要介绍本文会使用到的统计方法和机器学习算法、描述我们收集的关于伊利股份的数据、并简单介绍我们的研究思路。

2.1 SVARIMA 模型概述

SVARIMA 模型是一种时间序列模型，它是 ARIMA 模型的拓展。

ARIMA 模型，差分整合移动平均自回归模型，又称整合移动平均自回归模型（移动也可称作滑动），时间序列预测分析方法之一。ARIMA（p，d，q）中，AR 是"自回归"，p 为自回归项数；MA 为"移动平均"，q 为滑动平均项数，d 为

使之成为平稳序列所做的差分次数（阶数）。“差分”一词虽未出现在 ARIMA 的英文名称中，却是关键步骤。

ARIMA（p，d，q）模型是 ARMA（p，q）模型的扩展。ARIMA（p，d，q）模型可以表示为：

$$\left(1-\sum_{i=1}^{p}\phi_i L^i\right)(1-L)^d X_t = \left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t$$

其中 L 是滞后算子，$d \in \mathbb{Z}, d > 0$.

1980 年 Sims 提出向量自回归模型。这种模型采用多方程联立的形式，它不以经济理论为基础，在模型的每一个方程中，内生变量对模型的全部内生变量的滞后值进行回归，从而估计全部内生变量的动态关系。我们在 VARIMA 模型中加入季节因素就构成了 SVARIMA 模型。

在某些时间序列中，存在明显的周期性变化，这种周期是由于季节性变化（包括季度、月度、周度等变化）或其他一些固有因素引起的。这类序列称为季节性时间序列。譬如，一个地区的气温值序列（每隔一小时取一个观测值）中除了含有以天为周期的变化，还含有以年为周期的变化。在经济领域中，季节性时间序列更是随处可见，如季度时间序列、月度时间序列、周度时间序列等。处理季节性时间序列只用以上介绍的方法是不够的。描述这类序列的模型之一便是季节 ARIMA 模型(seasonal ARIMA model)，用 SARIMA 表示。

季节 ARIMA 过程是一般的 ARIMA 过程在季节时间序列模型中的推广。按照它们在均值和方差上的不同特征，可以分为确定性季节过程和季节单整过程。

2.2 神经网络算法概述

人工神经网络（ANN），简称神经网络（NN）或类神经网络，在机器学习和认知科学领域，是一种模仿生物神经网络（动物的中枢神经系统，特别是大脑）的结构和功能的数学模型或计算模型，用于对函数进行估计或近似。神经网络由大量的人工神经元联结进行计算。大多数情况下人工神经网络能在外界信息的基础上改变内部结构，是一种自适应系统。现代神经网络是一种非线性统计性数据建模工具。典型的神经网络具有以下三个部分：

结构指定了网络中的变量和它们的拓扑关系。例如，神经网络中的变量可以是神经元连接的权重和神经元的激励值。

激励函数大部分神经网络模型具有一个短时间尺度的动力学规则，来定义神经元如何根据其他神经元的活动来改变自己的激励值。一般激励函数依赖于网络中的权重（即该网络的参数）。

学习规则学习规则指定了网络中的权重如何随着时间推进而调整。这一般被看做是一种长时间尺度的动力学规则。一般情况下，学习规则依赖于神经元的激励值。它也可能依赖于监督者提供的目标值和当前权重的值。例如，用于手写识别的一个神经网络，有一组输入神经元。输入神经元会被输入图像的数据所激发。在激励值被加权并通过一个函数（由网络的设计者确定）后，这些神经元的激励值被传递到其他神经元。这个过程不断重复，直到输出神经元被激发。最后，输出神经元的激励值决定了识别出来的是哪个字母。

神经网络的构筑理念是受到生物（人或其他动物）神经网络功能的运作启发而产生的。人工神经网络通常是通过一个基于数学统计学类型的学习方法得以优化，所以人工神经网络也是数学统计学方法的一种实际应用，通过统计学的标准数学方法我们能够得到大量的可以用函数来表达的局部结构空间，另一方面在人工智能学的人工感知领域，我们通过数学统计学的应用可以来做人工感知方面的决定问题（也就是说通过统计学的方法，人工神经网络能够类似人一样具有简单的决定能力和简单的判断能力），这种方法比起正式的逻辑学推理演算更具有优势。

和其他机器学习方法一样，神经网络已经被用于解决各种各样的问题，例如机器视觉和语音识别。这些问题都是很难被传统基于规则的编程所解决的。

递归神经网络（RNN）是两种人工神经网络的总称。一种是时间递归神经网络，另一种是结构递归神经网络。时间递归神经网络的神经元间连接构成矩阵，而结构递归神经网络利用相似的神经网络结构递归构造更为复杂的深度网络。RNN 一般指代时间递归神经网络。单纯递归神经网络因为无法处理随着递归，权重指数级爆炸或消失的问题，难以捕捉长期时间关联；而结合不同的 LSTM 可以很好解决这个问题。

时间递归神经网络可以描述动态时间行为，因为和前馈神经网络接受较特定结构的输入不同，RNN 将状态在自身网络中循环传递，因此可以接受更广泛的时间序列结构输入。手写识别是最早成功利用 RNN 的研究结果。

2.3 数据说明

我们使用以下因子来拟合伊利的营业收入、营业成本、利润和股价。

表 1 因子列表

Factor	因子名称
1	进口数量:原奶(0401):当月值
2	新西兰:原奶产量
3	国际现货价:玉米
4	国际现货价:豆粕
5	玉米:产量
6	M2
7	预测平均值:CPI:当月同比
8	城镇居民人均消费性支出:累计同比
9	伊利股份:产量:液体乳
10	平均到岸价:苜蓿草:当月值
11	美国:牛:市场存栏量
12	进口数量:奶粉:当月值
13	进口金额:奶粉:当月值
14	存栏:奶牛:全国
15	产量:奶类:全国
16	产量:奶粉:中国
17	产量:牛奶:全国
18	人口数:全国
19	进口数量:种牛冻精
20	进口数量:苜蓿草
21	零售价:牛奶

为方便表述，后文将因子用数值标签代替，参考表格。

考虑到营业收入、营业成本、利润三个变量均为季度频，而股价为日频，因而本文预测模型分为两个档次，季度频为 Plate1，日频为 Plate2

2.4 研究思路

对目标进行分析，需要对伊利营业成本、营业收入、利润和股价多个因变量建立预测模型，此过程可以分为数据预处理，因子筛选与合成降维，模型搭建与展望四个部分。考虑到前三个变量均为季度频，而股价为日频，因而预测模型初步可以分为两个档次。

首先是对数据的预处理，预处理可以分为三个阶段：1、数据清洗；2、数据对齐；3、数据转换。

其次是对因子的筛选与合成降维：因子筛选分别采用了筛除低波动，单变量筛选，Lasso 正则化、随机 Lasso 和贪心算法等方式进行因子筛选。

然后是分别对两个档次进行预测模型的搭建：

一、考虑对营业成本、营业收入、利润建立预测模型：结合对实际数据的初步判断，发现伊利的营业成本、营业收入、利润具有很强的季节效应，拟建立具有季节效应的时间序列模型。前期，尝试利用深度学习神经网络建立预测模型，发现预测结果并不理想，分析原因很有可能是由于季节效应和过小的样本量所致，降低深度学习神经网络用来构造预测模型的优先级；同时由于数据具有强的前后时间效应，故暂时不考虑忽略时间效应的多元回归模型，因而对这个档次建立预测模型时，优先考虑自回归移动平均时间序列模型，同时考虑到不同时间序列的相关性和潜在有效信息，综合考虑为利用季节序列单整向量自回归移动平均时间序列 SVARIMA 建立预测模型。

二、考虑对伊利股价建立预测模型：结合对实际数据的初步判断，伊利的股价序列为日频，同时观察到伊利股价 5000+的样本量，考虑到机器学习在大样本量学习优化建立预测模型的优势，同时考虑到深度学习，这里拟采用长短期记忆神经网络 RNN-LST 来建立预测模型。

最后是对模型的展望，上述模型存在如下可以拓展的地方：在构建时间序列模型当中，可以考虑加入到其它一些外生变量作为因子，构造 SVARIMAX 时间

序列预测模型，同时后续可以考虑运用改进的 CNN 神经网络的方式建立预测模型等。

第三章 数据处理与模型建立

3.1 数据预处理

首先进行数据清洗。利用 python 进行对原始数据异常值、缺失值、非相关值、重复值的清洗，筛去缺失值，异常值，非相关数据，不一致数据和重复数据，保证数据的稳定性。

其次处理缺失值。常见的处理缺失值的方法有插值法，移动平均，均值回归等。考虑到所提供数据中基本均为不同发布频率的原始数据，为了保证数据的有效性和合理性，这里采用了近邻值的方法、分别以两个档次因变量数据为参考序列实现了数据的对齐。

同时，考虑到数据之间数量级有相当大的差异，为有效的构建预测模型，我们对因子、利润表数据和股价进行规范化和标准化的规整处理。在分别对归一化、规范化和 z 分位数标准化尝试进行数值变换的过程之后，发现 z 分位数标准化的预测效果更显著，因而采用 z 分位数标准化的方式进行数值转换。

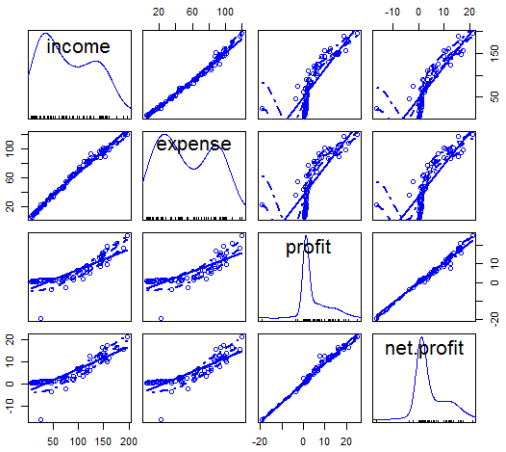

图 1 变量相关情况

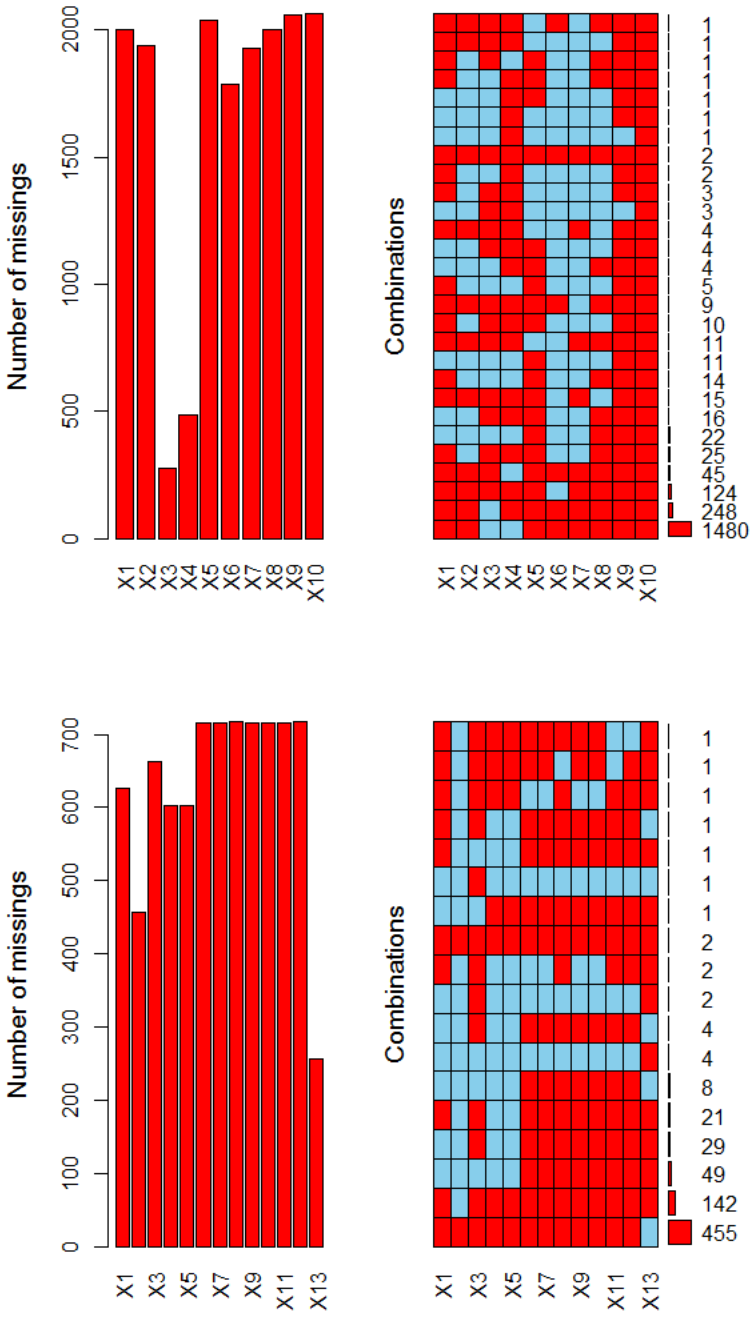
Number of missings
0
500
1000
1500
2000
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
Combinations
1
1
1
1
1
1
2
2
3
3
4
4
4
5
9
10
11
11
14
15
16
22
25
45
124
248
1480
Number of missings
0
100
200
300
400
500
600
700
X1
X3
X5
X7
X9
X11
X13
Combinations
1
1
1
1
1
1
1
2
2
2
4
4
8
21
29
49
142
455

图 2 缺失值情况

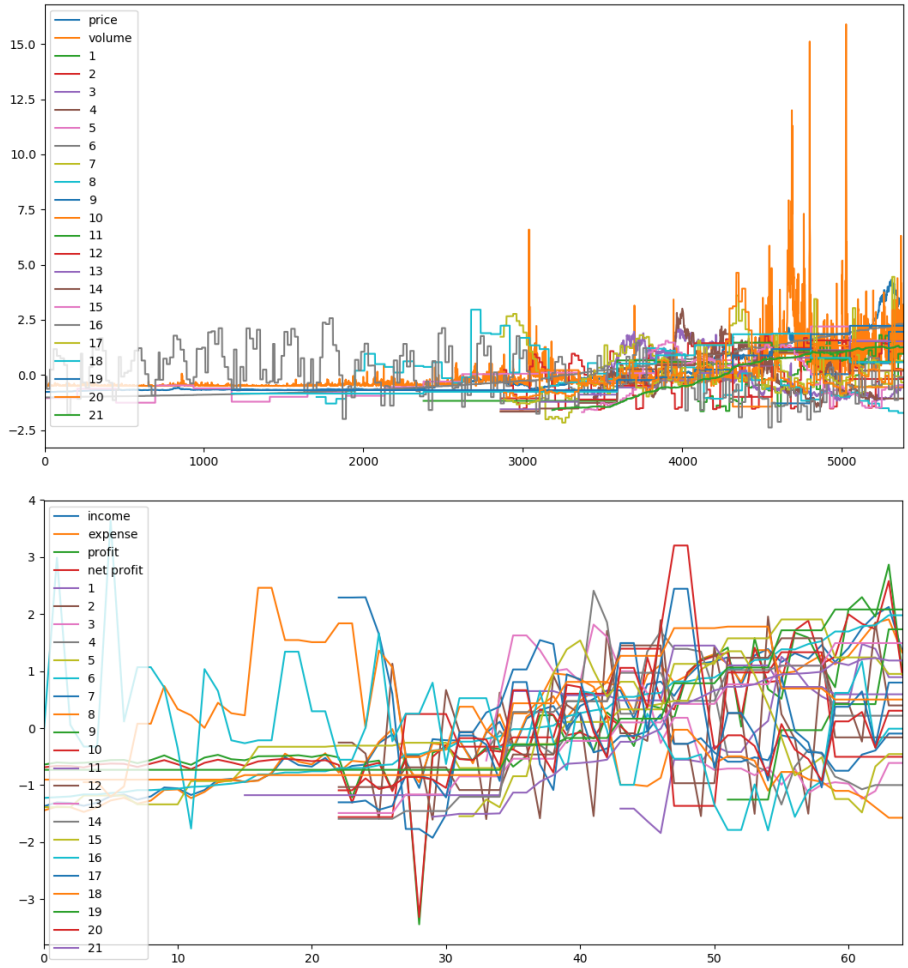

图 3 数据标准化前后

3.2 特征筛选

3.2.1 筛去取值变化幅度小的特征因子

下图为对 21 个因子的描述性统计数据，这里在于筛选出取值变化较小的特征因子，由于数据标准化之后可以对数据进行横向比较，发现 X9 的取值变化较小，因而初步筛除出因子 9.

表 2 描述性统计量

	X1	X2		X9	X10		X17	X18
n	5.385000e+03	5.385000e+03	n	5.385000e+03	5.385000e+03	n	5.385000e+03	5.385000e+03
mean	-4.081573e-18	7.730223e-18	mean	-2.265266e-14	-3.711597e-13	mean	9.126395e-17	1.598565e-17
stdev	4.887288e-01	6.818328e-01	stdev	3.909736e-01	6.846871e-01	stdev	6.818328e-01	8.842774e-01
skw	1.118778e-01	-1.981439e-01	skw	5.947631e-01	-2.381365e-01	skw	2.505465e+00	7.563217e-01
kurtosis	4.511937e+00	9.312519e-01	kurtosis	1.310610e+01	1.012726e+00	kurtosis	1.164121e+01	-5.883540e-01
	X3	X4		X11	X12		X19	X20
n	5.385000e+03	5.385000e+03	n	5.385000e+03	5.385000e+03	n	5.385000e+03	5.385000e+03
mean	5.833738e-16	-1.796795e-14	mean	1.649247e-14	1.605372e-13	mean	-2.497236e-15	2.211343e-17
stdev	6.021260e-01	6.004270e-01	stdev	7.473330e-01	6.846871e-01	stdev	8.842774e-01	6.818328e-01
skw	9.873633e-01	1.246849e+00	skw	-2.531693e-01	6.452078e-01	skw	1.343265e+00	3.316559e+00
kurtosis	1.953098e+00	4.226470e+00	kurtosis	-6.030925e-01	4.222801e-01	kurtosis	7.210068e-01	1.571696e+01
	X5	X6		X13	X14		X21	
n	5.385000e+03	5.385000e+03	n	5.385000e+03	5.385000e+03	n	5.385000e+03	
mean	-2.636030e-16	-1.379810e-14	mean	-2.990052e-15	-3.992329e-14	mean	4.122372e-17	
stdev	1.000000e+00	1.000000e+00	stdev	6.846871e-01	6.846871e-01	stdev	6.392325e-01	
skw	7.811962e-01	8.383939e-01	skw	3.866158e-02	-9.628696e-01	skw	-3.715211e-01	
kurtosis	-4.901558e-01	-6.370003e-01	kurtosis	7.624604e-01	9.362128e-01	kurtosis	7.473813e-01	
	X7	X8		X15	X16			
n	5.385000e+03	5.385000e+03	n	5.385000e+03	5.385000e+03			
mean	3.981936e-15	5.037279e-14	mean	-1.868327e-15	-5.413201e-16			
stdev	6.846871e-01	8.265199e-01	stdev	6.116137e-01	1.000000e+00			
skw	9.025773e-01	7.393425e-01	skw	-2.666069e-01	2.634209e-01			
kurtosis	4.961200e+00	1.434400e+00	kurtosis	1.235929e+00	-4.601947e-01			

3.2.2 单变量特征筛选

通过 Pearson 相关系数，互信息和最大信息系数，距离相关系数，多重共线性（修正 Frish 逐步回归），基于机器学习模型的特征排序，这五种方法的综合判别，根据综合结果初步筛除出因子 5，因子 12，因子 18，因子 19，因子 20，因子 21。

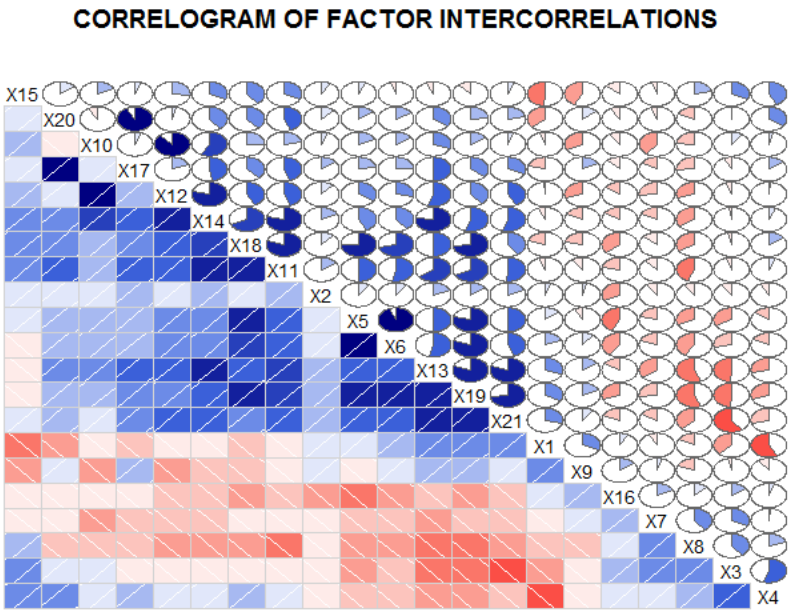

图 4 因子相关性

3.2.3 正则化特征筛选

优先采用 LASSO 的方法进行变量筛选，这里以 price 作为因变量用于演示通过计算出权重向量图 5，得到最优权重为 8.4*10^-1，对应图 6。最终，综合多个结果，筛除出权重接近于 0 的因子：因子 6，因子 12，因子 16，因子 19。

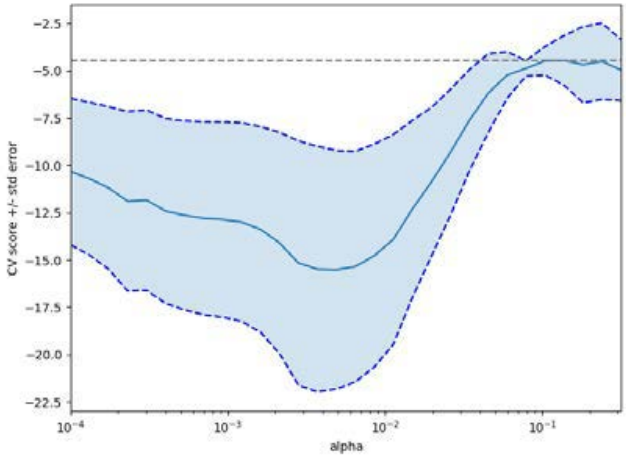

图 5 权重向量图

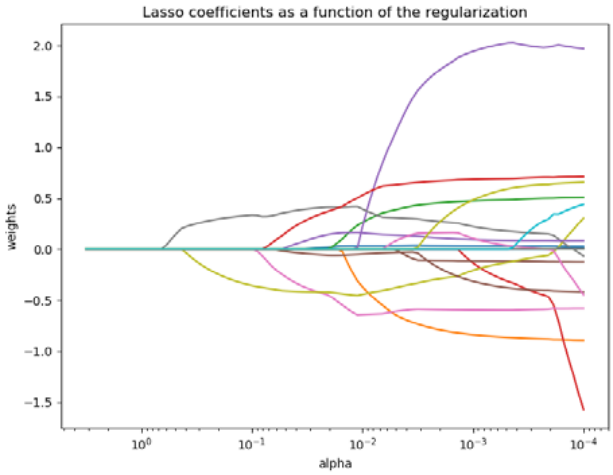

图 6 最优权重

3.2.4 随机森林特征筛选

我们采取随机森林机器学习算法，以平均减少不纯度作为特征选择值，以价格序列作为预测序列，最终得到 21 个因子各因子打分值如下：

表 3 随机森林特征筛选打分情况

Features sorted by their score:

[(0.8772, 5), (0.059, 1), (0.0254, 20), (0.0075, 2), (0.0067, 3), (0.0056, 14), (0.0055, 16), (0.0051, 6), (0.0032, 15), (0.0028, 19), (0.0012, 4), (0.0007, 7), (0.0001, 18), (0.0, 17), (0.0, 13), (0.0, 12), (0.0, 11), (0.0, 10), (0.0, 9), (0.0, 8)]

Note:其中数对第一个值表示因子打分值，第二个表示因子序数

这里拟初步筛除因子 8，因子 9，因子 10，因子 11，因子 12，因子 13，因子 17 因子打分值较低的序列

3.2.5 稳定性选择

采用稳定性选择的方法，利用随机 lasso 和随机逻辑回归实现，结果如下：

表 4 稳定性选择打分情况

Features sorted by their score:
[(1.0, 15), (1.0, 14), (1.0, 8), (1.0, 7), (1.0, 5), (1.0, 4), (1.0, 3), (1.0, 2), (1.0, 1), (0.99, 6), (0.985, 16), (0.715, 20), (0.685, 19), (0.12, 11), (0.1, 17), (0.095, 9), (0.05, 10), (0.035, 18), (0.035, 13), (0.02, 12)]

Note:其中数对第一个值表示因子打分值，第二个表示因子序数

根据结果拟筛除出因子 12

3.2.6 递归特征消除

表 5 递归特征消除打分情况

利用寻找最优特征子集的贪心算法将递归特征消除，得到因子打分结果如下：

Features sorted by their score:
[(1, 13), (2, 9), (3, 18), (4, 17), (5, 12), (6, 10), (7, 11), (8, 4), (9, 8), (10, 5), (11, 2), (12, 19), (13, 16), (14, 14), (15, 7), (16, 3), (17, 20), (18, 6), (19, 15), (20, 1)]

Note:其中数对第一个值表示因子打分值，第二个表示因子序数

根据结果拟筛除出因子 1 和因子 19

第四章 模型建立

模型的建立共分为两个阶段，分别是对季度频数据（营业收入、营业成本、利润）建立预测模型以及对日频数据（股价）建立预测模型

针对第一阶段，我们运用了季节序列单整向量自回归移动平均时间序列预测模型，即 SVARIMA。

针对第二阶段，我们建立了子模型二-长短期记忆神经网络 RNN-LST。

4.1 SVARIMA

4.1.1 模型构建

时间序列{X_t} 被称为周期为 S 的季节 ARIMA(p,d,q) $(P,D,Q)_s$ 过程，如果它的差分序列$Y_t = (1-B)^d(1-B^s)^D x_t$是一个平稳自回归滑动平均过程，即

$$\phi(B)\Phi(B^S)Y_t = \theta(B)\Theta(B^S)e_t$$

式中：B 表示向后差分算子; ϕ (z)、θ (z)、Φ (z)、Θ (z)为多项式，它们的阶数分别为 p、P、q、Q; e_t 服从均值为 0、方差为 σ^2 的正态分布，并且对任意的 k，

$Cov(e_t,e_{t-k})=0$．上述时间序列模型通常可以记为 SARIMA（p，d，q）（P，D，Q）S。

结合对实际数据的初步判断，发现伊利的营业成本、营业收入、利润具有很强的季节效应，分别建立具有季节效应的时间序列模型。

$$(1-B)(1-B^S)z_t = (I_k - \theta B)(I_k - \Theta B^S)a_t$$

同时由于样本量大小的限制（65），为保证矩阵的可逆性，建立该模型只能选取单因子和预测变量构建 sVARIMA 模型，这里分别采用前文优选的因子和预测因子配对构建预测模型。

在经历过模型阶数的判别之后，建立参数为 VARIMA(1,1,2)-Sorder(1,0,1)的 SVARIMA 预测模型。

4.1.2 结果输出

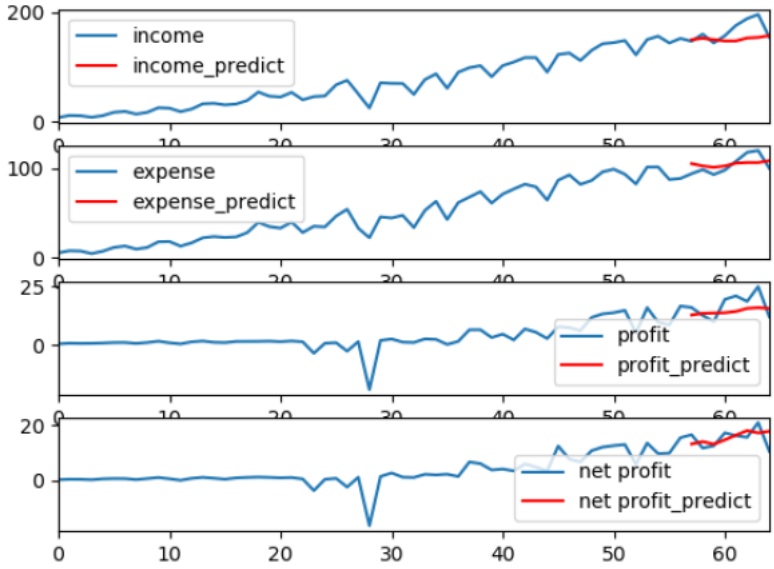

图 7 SVARIMA 预测结果

如图 7 所示，模型预测结果与实际数据趋势总体相符，但由于信息限制，数据的波动难以被很好的解释。

4.2 LSTM-RNN

4.2.1 模型构建

首先将高维时间序列（5382*23）转换为可供于监督学习的数据集（5382*92）

其次将数据集分为训练数据和测试数据集，用于预测（这里采用预测区间为 3 个月 92 天）

构造拟合模型：这里选取 MAE 作为损失函数和梯度下降的参数，构造 50 个神经元的隐藏层和 1 个神经元的输出层，设置每个神经元处理大小为 72，滞后期调整为 3，方法如下图所示

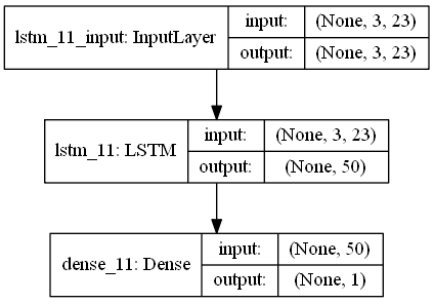

图 8 LSTM-RNN 模型构建

4.2.1 结果输出

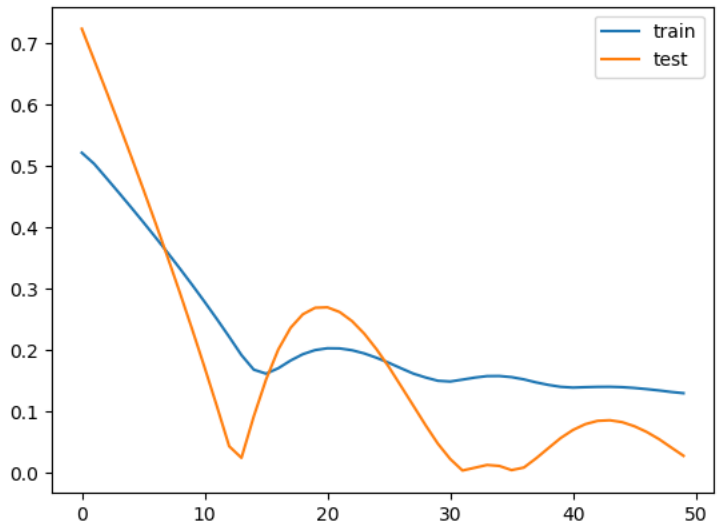

图 9 训练集及测试集表现

如图 9 所示，随着迭代次数的增加，学习效果逐渐改善。

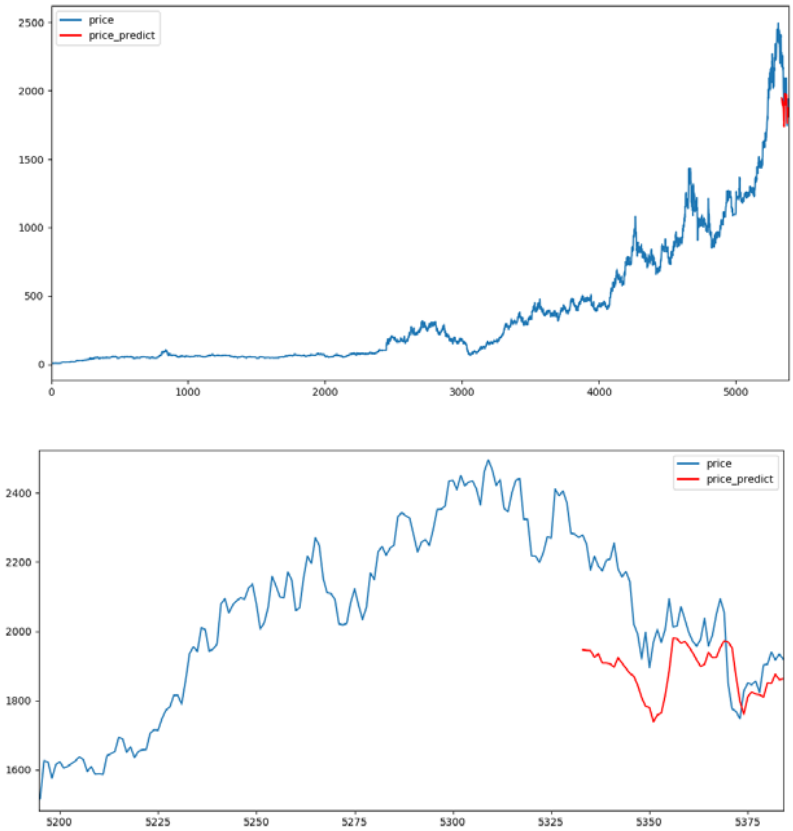

图 10 LSTM-RNN 预测结果

如上图所示，在时间序列层面上，模型能够较好的捕捉到股价的变动，效果理想。

第五章 模型评价与展望

5.1 模型评价

本文综合考虑到了数据的特殊性，分别通过时间序列和深度学习的方法建立预测模型。同时模型的建立综合利用了多种方法进行因子的筛选与合成

5.2 未来展望

后期可以考虑加入合理的外生变量 X 的方法来建立预测模型 sVARIMAX 同时也可以考虑到 BEKK、GARCH、ARCH 建立多元波动率模型；后期可以考虑 CNN 等深度神经网络的改进来建立预测模型。后续可自行研究添加可供参考因子。后续可以完善因子合成 PCA 等方式。

参考文献

[1] 股票定价理论 - MBA 智库百科. http://wiki.mbalib.com/wiki/%E8%82%A1%E7%A5%A8%E5%AE%9A%E4%BB%B7%E7%90%86%E8%AE%BA.
[2] 人工神经网络. 维基百科，自由的百科全书 (2018).
[3] 递归神经网络. 维基百科，自由的百科全书 (2017).
[4] 宗群.基于季节自回归单整移动平均模型的电梯交通流递归预测方法[J].天津大学学报, 2008(6)
[5] Ruey S.Tsay. Multivariable Time Series Analysis: With R and Financial Appli cations [M],2016

附录

项目说明文档及代码已开源，地址如下：

https://github.com/nkuzhengwt/Forecasting_Model_of_Seasonal_MultipleTimeSeries

详细代码如下：

```
#yili.py
from __future__ import print_function
print(__doc__)
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.linear_model import LassoCV
from sklearn.linear_model import Lasso
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from math import sqrt
from numpy import concatenate
```

```
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.utils.vis_utils import plot_model
from keras.layers import LSTM
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import RandomizedLasso
from sklearn.datasets import load_boston
class yili(object):
    def __init__(self,data):
        self.data = data
    def combine_income(self):
#         a = pd.read_excel('笔试题.xlsx',sheetname=[0,1,2,3])
        a = self.data
        a[2].drop(a[2].index[[0,-1,-2]],inplace=True)
        a[3].drop(a[3].index[[0,-1,-2]],inplace=True)
        a[2]=a[2].reset_index(drop=True)
        a[3]=a[3].reset_index(drop=True)
        a[0]['公告日期']=pd.to_datetime(a[0]['公告日期'])
        a[1]['日期']=pd.to_datetime(a[1]['日期'])
        a[2]['指标名称']=pd.to_datetime(a[2]['指标名称'])
        a[3]['指标名称']=pd.to_datetime(a[3]['指标名称'])
        #a[2] = a[2].fillna(method = 'ffill')
```

```
        #a[3] = a[3].fillna(method = 'ffill')
        b  =pd.merge(a[0],a[2],left_on='公告日期',right_on='指标名称',how='outer')
        c =pd.merge(a[0],a[3],left_on = '公告日期',right_on = '指标名称',how = 'outer')

        for j in range(len(b)):
            if (b['指标名称'].isnull())[j]:  # 如果为空即插值。
                b.loc[j,'指标名称'] = b.loc[j,'公告日期']
        b = b.sort_values(by='指标名称')
        b[b.columns[6:]] = b[b.columns[6:]].fillna(method='ffill')
        b=b[b['公告日期'].notnull()]
        b=b.reset_index(drop=True)

        for j in range(len(c)):
            if (c['指标名称'].isnull())[j]:  # 如果为空即插值。
                c.loc[j,'指标名称'] = c.loc[j,'公告日期']
        c = c.sort_values(by='指标名称')
        c[c.columns[6:]] = c[c.columns[6:]].fillna(method='ffill')
        c=c[c['公告日期'].notnull()]
        c=c.reset_index(drop=True)

        cols_to_use = c.columns.difference(b.columns)
        d        =        pd.merge(b,c[cols_to_use],left_index=True,right_index=True,how = 'outer')

        d.columns=['date','income','expense','profit','net profit','date2',\
                   '1','2','3','4','5',\
```

```
                    '6','7','8','9','10','11','12','13', \
                    '14','15','16','17','18','19','20','21']
        self.income = d
#         d.to_csv('income3.csv',index=False)
    def combine_price(self):
        a = self.data
        a[2].drop(a[2].index[[0,-1,-2]],inplace=True)
        a[3].drop(a[3].index[[0,-1,-2]],inplace=True)
        a[2]=a[2].reset_index(drop=True)
        a[3]=a[3].reset_index(drop=True)

        a[0]['公告日期']=pd.to_datetime(a[0]['公告日期'])
        a[1]['日期']=pd.to_datetime(a[1]['日期'])
        a[2]['指标名称']=pd.to_datetime(a[2]['指标名称'])
        a[3]['指标名称']=pd.to_datetime(a[3]['指标名称'])
        #a[2] = a[2].fillna(method = 'ffill')
        #a[3] = a[3].fillna(method = 'ffill')
        b  =pd.merge(a[1],a[2],left_on='日期',right_on='指标名称',how='outer')
        c  =pd.merge(a[1],a[3],left_on  = '日期',right_on  = '指标名称',how = 'outer')

        for j in range(len(b)):
            if (b['指标名称'].isnull())[j]:  # 如果为空即插值。
                b.loc[j,'指标名称'] = b.loc[j,'日期']
        b = b.sort_values(by='指标名称')
        b[b.columns[4:]] = b[b.columns[4:]].fillna(method='ffill')
        b=b[b['日期'].notnull()]
        b=b.reset_index(drop=True)
```

```
        for j in range(len(c)):
            if (c['指标名称'].isnull())[j]:  # 如果为空即插值。
                c.loc[j,'指标名称'] = c.loc[j,'日期']
        c = c.sort_values(by='指标名称')
        c[c.columns[4:]] = c[c.columns[4:]].fillna(method='ffill')
        c=c[c['日期'].notnull()]
        c=c.reset_index(drop=True)

        cols_to_use = c.columns.difference(b.columns)
        d         =         pd.merge(b,c[cols_to_use],left_index=True,
right_index=True,how = 'outer')
#         temp=d.copy()

d.columns=['date','price','volume','date2','1','2','3','4','5',\
                  '6','7','8','9','10','11','12','13', \
                  '14','15','16','17','18','19','20','21']
        self.price = d
#         d.to_csv('price3.csv',index=False)
    def plot_standardization(self):
        def is_numeric(s):
            try: float(s)
            except:
                return False
            else:
                return True

        a1= self.income
        a2= self.price
```

```
        #(a1['income']  -  a1['income'].min())/(a1['income'].max()  -
a1['income'].min())
        plt.figure(1)
        for i in range(len(a1.columns)):
        #    if is_numeric(a1[a1.columns[i]][0]):
        #                  a1[a1.columns[i]]  =(a1[a1.columns[i]]  -
a1[a1.columns[i]].min())/ \
        #        (a1[a1.columns[i]].max() - a1[a1.columns[i]].min())
        #        a1[a1.columns[i]].plot()
            if is_numeric(a1[a1.columns[i]][0]):
                a1[a1.columns[i]]        =(a1[a1.columns[i]]        -
a1[a1.columns[i]].mean())/ \
                a1[a1.columns[i]].std()
                a1[a1.columns[i]].plot()
        plt.legend()

        plt.figure(2)
        for i in range(len(a2.columns)):
        #    if is_numeric(a2[a2.columns[i]][0]):
        #                  a2[a2.columns[i]]  =(a2[a2.columns[i]]  -
a2[a2.columns[i]].min())/ \
        #        (a2[a2.columns[i]].max() - a2[a2.columns[i]].min())
        #        a2[a2.columns[i]].plot()
            if is_numeric(a2[a2.columns[i]][0]):
                a2[a2.columns[i]]        =(a2[a2.columns[i]]        -
a2[a2.columns[i]].mean())/ \
                a2[a2.columns[i]].std()
                a2[a2.columns[i]].plot()
        plt.legend()
```

```
        self.income1 = a1
        self.price2 = a2
#         a1.to_csv('income4.csv',index=False)
#         a2.to_csv('price4.csv',index=False)
    def lasso(self,data):
        a1=data
        a1=a1.dropna()
        y =a1['price'].values
        X=a1[a1.columns[5:27]].values
        #diabetes = datasets.load_diabetes()
        #X = diabetes.data[:150]
        #y = diabetes.target[:150]

        lasso = Lasso(random_state=0)
        alphas = np.logspace(-4, -0.5, 30)

        tuned_parameters = [{'alpha': alphas}]
        n_folds = 3

        clf = GridSearchCV(lasso, tuned_parameters, cv=n_folds,
refit=False)
        clf.fit(X, y)
        scores = clf.cv_results_['mean_test_score']
        scores_std = clf.cv_results_['std_test_score']
        plt.figure().set_size_inches(8, 6)
        plt.semilogx(alphas, scores)

        # plot error lines showing +/- std. errors of the scores
        std_error = scores_std / np.sqrt(n_folds)
```

```
        plt.semilogx(alphas, scores + std_error, 'b--')
        plt.semilogx(alphas, scores - std_error, 'b--')

        # alpha=0.2 controls the translucency of the fill color
        plt.fill_between(alphas, scores + std_error, scores -
std_error, alpha=0.2)

        plt.ylabel('CV score +/- std error')
        plt.xlabel('alpha')
        plt.axhline(np.max(scores), linestyle='--', color='.5')
        plt.xlim([alphas[0], alphas[-1]])

        #
##############################################################################
########
        # Bonus: how much can you trust the selection of alpha?

        # To answer this question we use the LassoCV object that sets
its alpha
        # parameter automatically from the data by internal cross-
validation (i.e. it
        # performs cross-validation on the training data it receives).
        # We use external cross-validation to see how much the
automatically obtained
        # alphas differ across different cross-validation folds.
        lasso_cv = LassoCV(alphas=alphas, random_state=0)
        k_fold = KFold(3)
```

```
        print("Answer to the bonus question:",
              "how much can you trust the selection of alpha?")
        print()
        print("Alpha parameters maximising the generalization score on different")
        print("subsets of the data:")
        for k, (train, test) in enumerate(k_fold.split(X, y)):
            lasso_cv.fit(X[train], y[train])
            print("[fold {0}] alpha: {1:.5f}, score: {2:.5f}".
                  format(k, lasso_cv.alpha_, lasso_cv.score(X[test], y[test])))
        print()
        print("Answer: Not very much since we obtained different alphas for different")
        print("subsets of the data and moreover, the scores for these alphas differ")
        print("quite substantially.")

        plt.show()
        plt.figure().set_size_inches(8, 6)

        #diabetes = datasets.load_diabetes()
        #X = diabetes.data[:150]
        #Y= diabetes.target[:150]
        a1=data
        a1=a1.dropna()
        y =a1['price'].values
        X=a1[a1.columns[5:27]].values
        coefs=[]
```

```
        n_alphas=100
        alphas=np.logspace(-4,0.5,n_alphas)

        for a in alphas:
            lasso=Lasso(alpha=a)
            lasso.fit(X,y)
            coefs.append(lasso.coef_)

        ax = plt.gca()

        ax.plot(alphas, coefs)
        ax.set_xscale('log')
        ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
        plt.xlabel('alpha')
        plt.ylabel('weights')
        plt.title('Lasso coefficients as a function of the
regularization')
        plt.axis('tight')
        plt.show()
        plt.legend()
    def LinearRegression(self,data):
        a1= data
        a1=a1.dropna()
        Y =a1['price'].values
        X=a1[a1.columns[5:27]].values
        names=list(range(1,22))

        #use linear regression as the model
```

```
        lr = LinearRegression()
        #rank all features, i.e continue the elimination until the last
one
        rfe = RFE(lr, n_features_to_select=1)
        rfe.fit(X,Y)

        print("Features sorted by their rank:")
        print(sorted(zip(map(lambda x: round(x, 4), rfe.ranking_),
names)))
    def Randomforest(self,data):
        a1 = data
        a1=a1.dropna()
        Y =a1['price'].values
        X=a1[a1.columns[5:27]].values
        names=list(range(1,22))
        #X = boston["data"]
        #Y = boston["target"]
        #names = boston["feature_names"]
        rf = RandomForestRegressor()
        rf.fit(X, Y)
        print("Features sorted by their score:")
        print(sorted(zip(map(lambda x: round(x, 4),
rf.feature_importances_), names),
                     reverse=True))
    def Randomlasso(self,data):
        a1= data
        a1=a1.dropna()
        Y =a1['price'].values
        X=a1[a1.columns[5:27]].values
```

```
        names=list(range(1,22))

        rlasso = RandomizedLasso(alpha=0.025)
        rlasso.fit(X, Y)

        print("Features sorted by their score:")
        print(sorted(zip(map(lambda x: round(x, 4), rlasso.scores_),
                         names), reverse=True))
    def Lstm(self,KIND):
        # convert series to supervised learning
        def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
          n_vars = 1 if type(data) is list else data.shape[1]
          df = DataFrame(data)
          cols, names = list(), list()
          # input sequence (t-n, ... t-1)
          for i in range(n_in, 0, -1):
              cols.append(df.shift(i))
              names += [('var%d(t-%d)' % (j+1, i)) for j in
range(n_vars)]
          # forecast sequence (t, t+1, ... t+n)
          for i in range(0, n_out):
              cols.append(df.shift(-i))
              if i == 0:
                names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
              else:
                  names += [('var%d(t+%d)' % (j+1, i)) for j in
range(n_vars)]
          # put it all together
          agg = concat(cols, axis=1)
```

```
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
def change(df,n):
    temp_name = df.columns[n]
    temp_data = df[temp_name]
    df.drop(df.columns[n],axis=1,inplace=True)
    df.insert(0, temp_name, temp_data)
    return df



def lstm(a1,change_num,max_num,train_num,kind):
    a1 = change(a1,change_num)
    dataset=a1
    values = dataset.values
    # integer encode direction
    #encoder = LabelEncoder()
    #values[:,4] = encoder.fit_transform(values[:,4])
    # ensure all data is float
    values = values.astype('float32')
    # normalize features
    scaled=values
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled = scaler.fit_transform(values)
    # specify the number of lag hours
    n_hours = 3
```

```
            n_features = len(dataset.columns)
            # frame as supervised learning
            reframed1 = series_to_supervised(scaled, n_hours, 1)
            reframed = reframed1
            print(reframed.shape)
            # drop columns we don't want to predict
            #reframed.drop(reframed.columns[[9,10,11,12,13,14,15]],
axis=1, inplace=True)
            # split into train and test sets
            values = reframed.values
            n_train_hours = train_num
            train = values[:n_train_hours, :]
            test = values[n_train_hours:, :]
            # split into input and outputs
            n_obs = n_hours * n_features
            train_X, train_y = train[:, :n_obs], train[:, -n_features]
            test_X, test_y = test[:, :n_obs], test[:, -n_features]
            print(train_X.shape, len(train_X), train_y.shape)
            # reshape input to be 3D [samples, timesteps, features]
            train_X = train_X.reshape((train_X.shape[0], n_hours,
n_features))
            test_X = test_X.reshape((test_X.shape[0], n_hours,
n_features))
            print(train_X.shape, train_y.shape, test_X.shape,
test_y.shape)

            # design network
            model = Sequential()
            model.add(LSTM(50, input_shape=(train_X.shape[1],
```

```
train_X.shape[2])))
            model.add(Dense(1))
            model.compile(loss='mae', optimizer='adam')
            plot_model(model,       to_file='model_plot_'+kind+'.png',
show_shapes=True, show_layer_names=True)

            # fit network
            history    =    model.fit(train_X,    train_y,    epochs=50,
batch_size=72,     validation_data=(test_X,     test_y),     verbose=2,
shuffle=False)
            # plot history
            plt.figure(1)
            plt.subplot(max_num,1,change_num+1)
            plt.plot(history.history['loss'], label='train')
            plt.plot(history.history['val_loss'], label='test')
            plt.legend()
            plt.show()
            # make a prediction
            yhat = model.predict(test_X)
            test_X          =          test_X.reshape((test_X.shape[0],
n_hours*n_features))
            # invert scaling for forecast
            inv_yhat = concatenate((yhat, test_X[:, (-n_features+1):]),
axis=1)
            inv_yhat = scaler.inverse_transform(inv_yhat)
            inv_yhat = inv_yhat[:,0]
            # invert scaling for actual
            test_y = test_y.reshape((len(test_y), 1))
            inv_y = concatenate((test_y, test_X[:, (-n_features+1):]),
```

```
axis=1)
            inv_y = scaler.inverse_transform(inv_y)
            inv_y = inv_y[:,0]
            # calculate RMSE
            rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
            print('Test RMSE: %.3f' % rmse)
            plt.figure(2)
            plt.subplot(max_num,1,change_num+1)
            dataset[dataset.columns[0]].plot(label=dataset.columns[0])
            plt.plot(range(len(dataset))[-
len(inv_yhat):],inv_yhat,color='red',label=dataset.columns[0]+'_predi
ct')
            plt.legend()
            plt.show()
            return rmse,inv_yhat,inv_y
        # load dataset
        #dataset = read_csv('pollution.csv', header=0, index_col=0)
        kind = KIND
        if kind == 'price':
            change_num,train_num = 1,5330
            a1 = self.price
        elif kind == 'income':
            change_num,train_num = 4,50
            a1 = self.income

        a1=a1.fillna(0)
        a1.drop('date',axis=1, inplace=True)
        a1.drop('date2',axis=1, inplace=True)
        a1=a1.reset_index(drop=True)
```

```
        total=pd.DataFrame(columns=['y_hat','y','rmse'])
        for i in range(change_num):
            rm,y_hat,y = lstm(a1,i,change_num,train_num,kind)
            temp1=pd.DataFrame({'y_hat':y_hat,'y':y,'rmse':rm})
            temp2=pd.DataFrame({'y_hat':[i],'y':[i],'rmse':[i]})
            total=pd.concat([total,temp1,temp2])
# main.py
from yili import *
import pandas as pd
if __name__ == '__main__':
    # read your own data
    data = pd.read_excel('data.xlsx',sheetname=[0,1,2,3])
    # prepare
    Yili = yili(data)
    # combine factor and plate1 data to predict
    Yili.combine_income()
    # cobing factor and plate2 data to predict
    Yili.combine_price()
    # plot and standardization
    Yili.plot_standardization()
    # feature selection
    Yili.lasso(Yili.price)
    Yili.LinearRegression(Yili.price)
    Yili.Randomforest(Yili.price)
    Yili.Randomlasso(Yili.price)
    # Lstm model to predict
    Yili.Lstm('income')
    Yili.Lstm('price')
library("MTS")
```

```
setwd("E:/Intern/CICC")
x<-read.table("income4.csv",header=TRUE,sep=',')
x<-subset(x,select=-c(date,date2))
mystats<-function(x,na.omit=FALSE){
  if(na.omit)
    x<-x[!is.na(x)]
  m<-mean(x)
  n<-length(x)
  s<-sd(x)
  skew<-sum((x-m)^3/s^3)/n
  kurt<-sum((x-m)^4/s^4)/n-3
  return(c(n=n,mean=m,stdev=s,skw=skew,kurtosis=kurt))
}
sapply(x,mystats)
summary(x)
x[is.na(x)]<-0
Yili<-x[c(1:60),c(1,10)]
Yili<-diffM(Yili)
plct<-VARMA(Yili,p=1,q=1)
VARMApred(plct,h=5)
plct<-VAR(Yili,p=2)
plct<-sVAR(Yili,p=2)
plct<-sVARMA(Yili,order=c(1,1,2),sorder = c(1,0,1),s=4)
```