统计与机器学习算法概要*

Yurk Lee†

2021/8/27

*仅供个人学习使用

†realyurk@gmail.com

目录

1 稳健性检验

1.1 什么是稳健性检验?

根据百度百科的解释，稳健性检验考察的是**评价方法**和**指标解释能力**的强壮性，也就是当改变某些参数时，评价方法和指标是否仍然对评价结果保持一个比较一致、稳定的解释。简单来说，当我们得出一个结论时，需要通过一系列方法来验证所得的结论是否可靠。当我们改变了一些条件或者假设发现所得结论依然不变，那么我们的结论就是稳健的，反之，所得结论有待商榷，我们需要找出使结论发生改变的原因并进行解释。

每篇文章根据自己的研究目的不同，稳健性检验的角度也会大不相同。比如当你的文章着重于研究方法的设计时，稳健性检验则应该更多关注于研究方法成立的前提条件和假设；而当你的文章数据处理时，则应该更多的关注于数据本身的稳健性。根据总结发现，常用的稳健性检验的角度包括**变量替换法，改变样本容量法，分样本回归法，补充变量法**等，下一章节中将按照该角度使用的频率从大到小进行排序并进行介绍。

当我们在课上学习到一个新方法时，老师会不断强调每个方法都有自己的假设和前提条件，而稳健性检验就是针对这些假设的。我们想要知道如果其中一个假设或者前提条件改变时，我们所得的结论是否依然可靠，这就是稳健性检验存在的意义。每当我们做稳健性检验时，我们应该思考以下问题:

- 我的研究假设是 A.
- 如果 A 不成立，那么我的结果 B 就可能出现有偏的估计 (可能估计值过高/过低/标准误过小/等等...)
- 我认为 A 在我的检验中可能不成立，因为 C 或者 D 是判断 A 是否成立的条件;
- 又如，D 是另外一种计量方法但是并没有 A 这个假设前提.
- 如果我们发现 A 不成立，那么我们则应该在稳健性检验中用 E 方法重新检验.

举一个简单的例子，假如我们现在准备研究政权的更替对于经济发展的影响，我们建立了一个简单的 OLS 回归模型将经济发展作为被解释变量，政权的更替作为核心解释变量进行估计:

- 我的分析假设是**扰动项均值独立于所有解释变量**，即变量外生，不受内部因素的影响，不存在遗漏变量的问题.
- **如果存在遗漏变量问题**，那么在回归中政权的更替这一变量的估计值就会过高或过低 (取决于遗漏了哪些变量)
- 我认为我们这个分析中存在**遗漏变量问题**，因为政权的更替通常会伴随着暴力事件的增加，而暴力事件的增加则会影响经济的发展，**所以暴力事件是我们在随机扰动项中没有控制的变量.**
- 那么，**增加暴力事件这一变量作为控制变量**是我可以进行的稳健性检验之一.
- 如果我们发现，增加了这一控制变量之后，使得我的结果与原先的结果完全不同，那么我们之前的结果则是不稳健的，我们应该加入这一变量进行重新估计.

本例中所提及的稳健性检验方法就是我们下文将要介绍的「补充变量法」，下面我们将系统地阐述这些变量方法.

1.2 变量替换法

在我们进行分析时，常常会选择自己最熟悉或者偏好的方法测量一个变量，而实际上一个变量的测量方法有很多种，我们根据以往文献研究或者依照自己数据可获得性选择的测量方法往往无法保证结论的可靠性。因此，在文献中，作者都会将变量替换法作为稳健性检验的方法之一，而在我们的统计中，变量替换法更是稳居检验角度第一名。变量替换法包括：**替换因变量，替换主要自变量**以及**放宽变量条件**等角度。

1.2.1 替换因变量

周京奎 (2019) 在研究农业生产率和农村家庭的人力资本积累关系时发现随着农业生产率提高，农村家庭倾向于进行教育投资，进而提升了家庭人力资本积累。在本文中作者首先采用家庭教育支出和家庭学杂费支出来衡量教育投资。在随后的稳健性检验章节中，作者将被解释变量替换为家庭教育支出占当年家庭收入的比例，考察农业生产率对教育支出占比的影响，进一步验证了农业生产率对人力资本投资影响的稳健性。

类似的文章可参考刘畅 (2017) 研究子女外出务工对农村父母身心健康的影响的文章，其中考虑到健康的多维性，采用了另外 6 个健康指标进行稳健性检验。

谭远发 (2015) 研究父母政治资本如何影响子女工资溢价的影响时，考虑到实际工资与保留工资正相关，因此将正文中子女的实际工资替换为保留工资进行稳健性检验。

李春涛 (2020) 研究金融科技发展对企业创新的影响时将企业的专利申请数量作为反映了企业的创新产出水平的衡量标准之一，随后作者进一步运用企业研发支出总额占销售收入的比例更替企业创新的度量指标进行稳健性检验。

此外孟美侠 (2019) ；罗勇根 (2019) ；陈强远 (2019) ；顾夏铭 (2018) 等都采用了替换因变量的方法进行了检验。

这里需要注意的一点是，除了替换因变量，学者有时还会对因变量进行一些修正，比如王雄元 (2019) 在检验国际贸易增加如何影响企业创新行为时考虑到未取自然对数的专利申请量数据为离散型变量，且其分布中存在大量 0 值，可能不符合正态分布的假定，因此采用泊松模型回归处理被解释变量非正态分布问题。

1.2.2 替换自变量

蔡晓慧 (2016) 在研究地方政府基础设施和企业技术创新关系时，正文部分讨论中使用的地方政府基础设施的数据来自于金戈 (2016) 估算的省级基础设施资本存量数据，而在稳健性检验中采用了地级市市辖区道路密度代表基础设施资本存量。因为道路交通是重要的基础设施，也是企业通过扩大市场规模取得规模经济的前提，道路交通的密度在一定程度上也反应了基础设施的基本存量。

1.2.3 放宽因变量或自变量条件

除了替换自变量与因变量外，学者有时还会对因变量或自变量的选择条件进行放宽，例如陈仕华 (2015) 在研究国企高管政治晋升对企业并购行为的影响时，对被解释变量的衡量主要是基于董

事长或总经理是否调任政府部门职位来判定高管政治晋升，考虑到董事长或总经理升任集团层面的董事长或总经理，或者升任集团层面的党委或党组书记时，国企高管的行政级别也得到了提升，因此在稳健性检验部分借鉴王曾等 (2014) 的测量方法，将高管职位变更去向出现以下情况时均视为晋升：平级或者更高级别的政府部门职位、集团层面的董事长或总经理、集团层面的党委或党组书记。以此替代变量进行测试。

上文中，我们介绍了稳健性检验的概念，目的以及常用的一个角度 (变量替换法) ，这篇文章我们将继续介绍稳健性检验的其他角度。从上篇推文可以看出，有些文章出现了不止一次，这说明，每一个稳健性检验的方法都是不是独立存在的，在一篇文章中学者可以根据自己的需要可以选择多个稳健性检验的方法，比如罗勇根 (2019) 在研究空气污染、人力资本流动与创新活力的关系一文中，一共采用了 8 种方法从多个维度来检验自己文章的稳健程度。

我们需要注意的是，稳健性检验的意义在于我们需要保证，文章得出的结论不会根据现在使用的数据的变化而发生巨大的变化，比如当其他人使用了一份相似的数据，或者当本文数据的样本量发生不同时，你的结论依然成立，这才能保证结论的可靠性。

1.3 补充变量法

在上文讲述稳健性检验时，我们曾举到一个例子，当探讨政权的更替对于经济发展的影响时，我们会产生遗漏变量的问题，而遗漏变量问题是我们大多数研究中都会遇到的问题，我们只能尽可能多的在模型中加入我们能想到的以及之前文献研究过的对我们结果可能产生影响的变量。因此，**控制变量法**和之前的**变量替换法**几乎成为每篇文献中都会使用到的稳健性检验方法。

1.3.1 加入遗漏变量

除了前文所举的例子以外，梁斌 (2020) 在探讨失业保险金对失业者求职努力的影响时，将失业者在日志日搜寻工作的小时数作为因变量，失业者领取到的失业保险金作为自变量，并控制了个体特征变量以及家庭特征变量，加入了省份虚拟变量后，在稳健性检验部分提出，失业保险金对失业者来说是确定性的收入，因此本文预期厌恶风险的失业者 (risk-aversion) 更可能领取失业保险金，也更可能为了日后稳定的收入而积极寻求工作，因此又将风险这一变量纳入了考量。

1.3.2 加入各类虚拟变量

需要注意的是，加入遗漏变量有时不仅仅指加入更多的变量，也包括控制其他层面的固定效应，比如施炳展 (2020) 在研究互联网对制造业企业分工水平的影响时提到，在前文中作者只控制了年份固定效应和企业固定效应，虽然大多数企业并不会更换省份和行业，但是这种可能性是客观存在的，因此如果不加入省份和行业固定效应，有可能遗漏省份和行业层面不随时间改变的重要变量，从而使估计结果有偏和不一致。为了避免这一问题，作者在保留年份和企业固定效应的基础上，进一步加入了省份和行业固定效应。

1.4 分样本回归法

由于不同的样本对于所得的结果具有不同的敏感性，因为在稳健性检验时，也常常进行分样本回归，常见的分类方法用按照人口规模分类，按照地理位置分类，按照城乡分类，按照性别不同分类等等。

比如，刘怡 (2017) 在研究婚姻匹配对代际流动性的影响时提出婚姻匹配是中国代际传递的重要机制，尤其是对女性而言，父代收入通过婚配市场作用于子代配偶的个人收入，形成代际传递，影响子代家庭收入。在稳健性检验中，作者根据子代的城乡分布，将子代样本划分为城镇和乡村样本，比较分析城镇和乡村地区的代际流动性及其婚姻匹配机制在代际传递中的影响，结果发现，城镇地区多依赖于婚姻匹配机制，而农村地区侧重于人力资本投资。

类似的分样本回归方法，可以参考杨仁发 (2013) 研究产业集聚与地区工资差距之间的内在联系的文章；蔡晓慧 (2016) 研究地方政府基础设施和企业技术创新关系的文章；刘畅 (2017) 研究子女外出务工对农村父母身心健康的影响的文章；申广军 (2017) 研究减税对中国经济的影响文章等。

1.5 调整样本期

当我们在所得的整个数据集范围内进行分析时，常常会发现改变不同的时间段，得到的结论可能会完全不同。也许某一结论在某一时间段内得到的结果符合我们的预期，而当我们往后退 10 年，或者往前推 10 年再次回归，就会发现得到的结论完全不同！因此，选择正确的研究时间段也显得十分重要。在稳健性检验中，我们可以通过扩宽时间长度或者缩短时间长度来检验我们的结论。

1.5.1 扩展时间窗口

仇童伟 (2019) 在研究宗族代理人对村庄地权变更的影响时在第一个稳健性检验方法中提到，村庄的丧葬习俗表征了社区开放程度，在原文中采用了 2012-2014 的数据，而在稳健性检验中补充采用 1990-2014 年村庄丧葬习俗进行了处理。因为与仅采用 2012-2014 年丧葬习俗相比，采用 6 个时期的丧葬习俗可以规避单一时期测量造成的误差。类似的文章还包括朱晓文 (2019) 研究家族企业代际传承的文章中。

为了探讨长期的影响，除了扩展时间窗口外，陈冬华 (2018) 在研究产业政策与股价同步性的关系中提到，产业政策作为一种国家级政策，每五年发布一次，影响周期为五年。因此，作为一种长期政策，其对企业的影响可能存在长期性，文章的研究区间应该扩展至全年度而非短时间区间范围。基于此，参考错层事件双重差分方法，文章进一步探究了国家产业政策影响股价同步性的长期表现。

1.5.2 缩短时间窗口

李卫兵 (2019) 在研究空气污染对企业生产率的影响时在稳健性检验部分提到该文选定的样本期为 1998-2013 年，而大部分基于中国工业企业数据库进行研究的文献主要利用 1998-2007 年的企业数据，虽然该文对某些缺失的数据根据相关的会计准则进行了补齐处理，为避免处理后的数据干扰实证结果，作者将样本调整为 1998-2007 年，并重新进行 RD 估计。

缩短时间窗口的另一个好处是可以排除其他政策的影响，比如王雄元 (2019) 在研究“一带一路”如何影响企业创新行为的研究中提到，中国于 2013 年正式提出“一带一路”倡议，因此在样本仅保留 2013 年及以后开通“中欧班列”的样本有助于将本文的研究统一置于“一带一路”倡议的背景下，排除可能的其他政策干扰。(注：另一种排除同时期其他政策的影响的影响是通过控制同时期政策带来的影响，比如齐绍洲 (2018) 在研究排污权交易试点政策是否诱发了企业绿色创新文章时提到，排污费征收政策与排污权交易试点政策并行，我们可以通过需要控制排污费征收政策对企业绿色创新的影响，进一步提炼排污权交易试点政策对企业绿色创新的因果关系。)

1.5.3 滚动窗口法

陈冬华 (2018) 在研究产业政策对股价同步性影响文章中提出，产业政策的影响是一个循序渐进的过程，因此在稳健性检验部分基于滚动窗口的实证研究方法对产业政策进行了动态研究。

1.6 改变样本容量法

当我们选择好了时间之后，同时也要确定我们的样本是否最能体现我们所研究的问题，同时样本中有没有极端值会影响我们的结果。因此，在稳健性检验中，我们需要将个别离群值剔除，或者在样本中选择最适合我们研究目的样本来检验我们的结论是否依然稳健。

1.6.1 选择子样本

鞠雪楠 (2020) 在研究跨境电商平台克服了哪些贸易成本时提出在跨境电商出口贸易中，中国向各个国家 (地区) 出口的分布并不均衡。其中，美国是中国最大的出口目的地; 中国香港和新加坡是全世界重要的转口贸易地区，中国向这个两个地区的出口可能也有转而向其他国家出口。为了确保实证分析的结论不受特定国家 (地区) 和转口贸易的影响，本文给出了剔除这三个国家以及地区的样本之后的实证分析结果。

1.6.2 缩尾处理

在处理离群值时，我们要进行缩尾处理，陈强远 (2019) 在研究中国技术创新主要激励政策对企业技术创新质量和数量的影响时提到，由于控制变量如资产收益率与负债比率的测算存在极端值，尽管上文已对资产收益率与负债比率进行了 5% 分位上双边缩尾。但为了进一步验证前文结论的稳健性，接下来本文对企业的资产收益率与负债比率进行了 1% 分位上双边缩尾处理。

1.6.3 扩充样本容量

除了剔除部分样本进行回归之外，我们依然可以通过增加样本来进行稳健性检验。比如原文中只采用了省会城市进行分析，在稳健性检验部分则可以将样本扩大到所有地级市城市，这一方法有时也被称为降低数据维度。

比如李卫兵 (2019) 在研究空气污染对企业生产率的影响时提到，本文提取的 PM2.5 排放浓度来源于城市层面，同时由于大样本选择下更易带来显著的回归结果，为了证明回归结果的准确性，我们参考江艇等 (2018) 的处理方法计算出城市层面的 TFP，将区域层面的数据降低至城市层面。(注：除了降低数据维度，我们同样可以提高数据维度，比如铁瑛 (2019) 在人口结构变动的影响时多个个体维度进行调整，分别加总至企业维度和城市维度进行稳健性分析)

1.7 内生性问题

内生性问题是我们每个文章都要考虑到的问题，施炳展 (2020) 在分析互联网对中国制造业企业分工水平的影响时将大部分稳健性检验的篇幅都留给了内生性问题，可见内生性问题对我们研究的重要性。在处理内生性问题时，我们通常采用以下几种方法进行稳健性检验：

1.7.1 工具变量法

工具变量是解决内生性问题的一个重要方法，比如施炳展 (2020) 选择了中国建国初期各省份人均函件数量作为省份层面企业互联网普及率的工具变量，选择一个合适的工具变量可以对整个研究都有重要的影响，但同时也是十分困难的，我们可以通过大量的文献阅读积累来选择最合适本文研究的工具变量。

类似的利用工具变量克服反向因果关系的文献可以参考蔡栋梁 (2018)；周京奎 (2019)；梁斌 (2020)；刘启仁 (2020)；张龙鹏 (2016)；罗勇根 (2019)

1.7.2 加入滞后变量

部分研究也会将自变量的滞后一期或者两期变量纳入模型中来解决内生性问题，比如孙传旺 (2019) 在研究交通基础设施与城市空气污染的关系时除了控制核心解释变量的内生性偏误，我们还担心其他控制变量也可能存在潜在的内生性问题。为了检验结果稳健并排除这一种担忧，将其他所有控制变量滞后一期；黄健柏 (2015) 到工业用地价格扭曲对企业过度投资的影响可能存在更长的时滞效应，把回归模型中的工业用地价格扭曲程度变量替换为滞后两期项，重新进行回归分析；李春涛 (2020) 考虑到创新投入也是影响专利产出的重要因素，本文在控制变量中加入企业创新投入的指标，并采用研发支出总额占销售收入之比来度量。由于创新投入对创新产出的影响具有时滞性，本文使用滞后一期的创新投入指标。

1.7.3 样本自选择问题

陈强远 (2019) 在研究中国技术创新主要激励政策对企业技术创新质量和数量的影响中提到，高新技术企业认定等技术创新激励政策可能存在自选择问题，即企业整体绩效较好的企业更容易享受优惠政策，这可能导致估计结果存在偏误。为了解决这一问题，文章采用 Heckman 两步法进行了稳健性检验。类似的文章包括蔡晓慧 (2016)；周颖刚 (2019) 等。

注：因为内生性问题十分重要，也有一些文章不将其作为稳健性检验的一部分，而是作为正文当中的一部分，比如高晶晶 (2019)；韩永辉 (2017)；余吉祥 (2019)

1.8 其它方法

1.8.1 验证前提条件

正如前文提到，稳健性检验就是为了检验回归方法中的前提条件是否满足，比如吕越 (2019) 在采用双重差分法研究“一带一路”倡议的投资对对外投资的影响时检验了 DID 的方法成立的条件，包括安慰剂检验，平行趋势检验等等，类似的文章周茂 (2019)；朱晓文 (2019)；梁斌 (2020)；陈冬华 (2018)

同样李卫兵 (2019) 也在使用 RD 估计时，辅助进行了 RD 检验的有效性检验。；类似文章还有梁若冰 (2016)

1.8.2 模型替换法

在上文中提到的蔡晓慧 (2016) 这篇文章中，作者依次在正文中采用线性概率模型进行研究后，在稳健性检验部分又依次采用 Logit 模型、Probit 模型进行估计基础设施对企业是否投入研发的影响；同样施炳展 (2020) 考虑到线性回归模型潜在的模型设定偏误，以面板 Tobit 模型替换线性回

归模型后重新进行了回归；李春涛 (2020) 认为本文使用的专利数量有大量的零值，存在截尾数据的特征，因此使用 Tobit 模型进一步检验金融科技发展对企业创新的影响；祝树金 (2020) 用断点回归能较好的识别因果关系，这里使用这种方法对前文的 DID 回归进行稳健性检验。

1.8.3 更换新的数据源

何兴强 (2019) 在探讨房价收入比对家庭消费房产财富效应的影响时，为了增强研究结论的稳健性，分别使用了调查数据、宏观数据、和不同的家庭调查数据重新估计本文的主要回归。这种方法对于数据的要求较高，因此使用频率较低.

2 数据的统计描述和可视化分析

数理统计研究的对象是受随机因素影响的数据，以下数理统计就简称统计，统计是以概率论为基础的一门应用学科。

数据样本少则几个，多则成千上万，人们希望能用少数几个包含其最多相关信息的数值来体现数据样本总体的规律。描述性统计就是搜集、整理、加工和分析统计数据，使之系统化、条理化，以显示出数据资料的趋势、特征和数量关系。它是统计推断的基础，实用性较强，在统计工作中经常使用。

面对一批数据如何进行描述与分析，需要掌握参数估计和假设检验这两个数理统计的最基本方法。

我们将用 Matlab 的**统计工具箱 (Statistics Toolbox)** 来实现数据的统计描述和分析。

2.1 统计的基本概念

总体是人们研究对象的全体，又称母体，如工厂一天生产的全部产品 (按合格品及废品分类)，学校全体学生的身高。

总体中的每一个基本单位称为个体，个体的特征用一个变量 (如 x) 来表示，如一件产品是合格品记 $x=0$ ，是废品记 $x=1$; 一个身高 170(cm) 的学生记 $x=170$ 。

从总体中随机产生的若干个个体的集合称为**样本**，或子样，如 n 件产品，100 名学生的身高，或者一根轴直径的 10 次测量。实际上这就是从总体中随机取得的一批数据，不妨记作 $x_1, x_2, \cdots, x_n, n$ 称为**样本容量**。

简单地说，**统计的任务是由样本推断总体**。

一组数据 (样本) 往往是杂乱无章的，做出它的频数表和直方图，可以看作是对这组数据的一个初步整理和直观描述。

将数据的取值范围划分为若干个区间，然后统计这组数据在每个区间中出现的次数，称为**频数**，由此得到一个频数表。以数据的取值为横坐标，频数为纵坐标，画出一个阶梯形的图，称为直方图，或频数分布图。

假设有一个容量为 n 的样本 (即一组数据)，记作 $x=(x_1, x_2, \cdots, x_n)$ ，需要对它进行一定的加工，才能提出有用的信息，用作对总体 (分布) 参数的估计和检验。**统计量**就是加工出来的、反映样本数量特征的函数，它不含任何未知量。

下面我们介绍几种常用的统计量。

1. 表示位置的统计量—算术平均值和中位数

   算术平均值 (简称均值) 描述数据取值的平均位置，记作 $\bar{x}$,

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

   中位数是将数据由小到大排序后位于中间位置的那个数值。Matlab 中 mean(x) 返回 x 的均值，median(x) 返回中位数。

2. 表示变异程度的统计量—标准差、方差和极差

   标准差 s 定义为

$$s = [\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2]^{\frac{1}{2}}$$

   它是各个数据与均值偏离程度的度量，这种偏离不妨称为变异。

   方差是标准差的平方 s^2 。

   极差是 $x = (x_1, x_2, \cdots, x_n)$ 的最大值与最小值之差。

   Matlab 中 std(x) 返回 x 的标准差，var(x) 返回方差，range(x) 返回极差。

   你可能注意到标准差 s 的定义 (2) 中，对 n 个 (xi − x) 的平方求和，却被 (n −1) 除，这是出于无偏估计的要求。若需要改为被 n 除，Matlab 可用 std(x,1) 和 var(x,1) 来实现。

3. 中心矩、表示分布形状的统计量—偏度和峰度

   随机变量 x 的 r 阶中心矩为 $E(x-Ex)^r$。

   随机变量 x 的偏度和峰度指的是 x 的标准化变量 $(x-Ex)/\sqrt{Dx}$ 的三阶中心矩和四阶中心矩:

$$\nu_1 = E[(\frac{x - E(x)}{\sqrt{D(x)}})^3] = \frac{E[(x - E(x))^3]}{(D(x))^{3/2}}$$

$$\nu_2 = E[(\frac{x - E(x)}{\sqrt{D(x)}})^4] = \frac{E[(x - E(x))^4]}{(D(x))^2}$$

偏度反映分布的对称性，ν_1 >0 称为右偏态，此时数据位于均值右边的比位于左边的多;ν_1 <0 称为左偏态，情况相反; 而 ν_1 接近 0 则可认为分布是对称的。

峰度是分布形状的另一种度量，正态分布的峰度为 3，若 2 比 3 大得多，表示分布有沉重的尾巴，说明样本中含有较多远离均值的数据，因而峰度可以用作衡量偏离正态分布的尺度之一。

Matlab 中 moment(x,order) 返回 x 的 order 阶中心矩，order 为中心矩的阶数。skewness(x) 返回 x 的偏度，kurtosis(x) 返回峰度。

在此我想讨论一下为什么标准差的定义中除以的项数为 $n-1$ 而不是 n，为什么说 n 是用于总体估计。

直观来说，样本的自由度为 $n-1$ 是因为我们减去了均值，导致样本数少 1，而当估计总体时，$(n-1) \sim n$ 是几乎相等的，所以当用 n 作为除数时意味着我们对整体进行估计。当然，一组数据的均值可能并不在其中，因此这种说法存在很多问题，下面我们用公式来推导一下。

如果强制采用 n 作为样本的估计会出现什么问题? 这会导致我们得到的结果是原样本的有偏

估计，假设 $E(S_1^2)$ 为除以 n 的标准差，那么我们有:

$$\begin{aligned}
E\left(S_1^2\right) &= \frac{1}{n}\sum_{i=1}^{n} E\left(\left(X_i-\bar{X}\right)^2\right) = \frac{1}{n}E\left(\sum_{i=1}^{n}\left(X_i-\mu+\mu-\bar{X}\right)^2\right) \\
&= \frac{1}{n}E\left(\sum_{i=1}^{n}\left(\left(X_i-\mu\right)^2-2\left(X_i-\mu\right)\left(\bar{X}-\mu\right)+\left(\bar{X}-\mu\right)^2\right)\right) \\
&= \frac{1}{n}E\left(\sum_{i=1}^{n}\left(X_i-\mu\right)^2-2\sum_{i=1}^{n}\left(X_i-\mu\right)\left(\bar{X}-\mu\right)+n(\bar{X}-\mu)^2\right) \\
&= \frac{1}{n}E\left(\sum_{i=1}^{n}\left(X_i-\mu\right)^2-2n(\bar{X}-\mu)(\bar{X}-\mu)+n(\bar{X}-\mu)^2\right) \\
&= \frac{1}{n}E\left(\sum_{i=1}^{n}\left(X_i-\mu\right)^2-n(\bar{X}-\mu)^2\right) \\
&= \frac{1}{n}\left(\sum_{i=1}^{n}E\left(\left(X_i-\mu\right)^2\right)-nE\left((\bar{X}-\mu)^2\right)\right) \\
&= \frac{1}{n}(n\operatorname{Var}(X)-n\operatorname{Var}(\bar{X})) \\
&= \operatorname{Var}(X)-\operatorname{Var}(\bar{X}) = \sigma^2-\frac{\sigma^2}{n} = \frac{n-1}{n}\sigma^2,
\end{aligned}$$

而 $\frac{n-1}{n}\sigma^2$ 并不等于 σ^2，因此我们需要调整系数才使得我们的估计为无偏估计。

统计量中最重要、最常用的是均值和标准差，由于样本是随机变量，它们作为样本的函数自然也是随机变量，当用它们去推断总体时，有多大的可靠性就与统计量的概率分布有关，因此我们需要知道几个重要分布的简单性质。

2.2 统计中几个重要的概率分布

随机变量的特性完全由它的 (概率) 分布函数或 (概率) 密度函数来描述。设有随机变量 X，其分布函数定义为 $X \leqslant x$ 的概率，即 $F(x) = P\{X \leqslant x\}$。若 X 是连续型随机变量，则其密度函数 $p(x)$ 与 $F(x)$ 的关系为

$$F(x) = \int_{-\infty}^{x} f(x)\,\mathrm{d}x.$$

上 α 分位数是下面常用的一个概念，其定义为: 对于 $0 < \alpha < 1$，使某分布函数 $F(x) = 1-\alpha$ 的 x，称为这个分布的上 α 分位数，记作 x_α 。

统计中几个重要的概率分布:

- **正态分布**

正态分布随机变量 X 的密度函数曲线呈中间高两边低、对称的钟形，期望 (均值) $EX = \mu$，方差 $DX = \sigma^2$，记作 $X \sim N(\mu, \sigma^2)$，σ 称均方差或标准差，当 $\mu = 0, \sigma = 1$ 时称为标准正态分布，记作 $X \sim N(0, 1)$。正态分布完全由均值 和方差 σ^2 决定，它的偏度为 0，峰度为 3。正态分布可以说是最常见的 (连续型) 概率分布，成批生产时零件的尺寸，射击中弹着点的位置，仪器反复量测的结果，自然界中一种生物的数量特征等，多数情况下都服从正态分布，这不仅是观察和经验的总结，而且有着深刻的理论依据，即在大量相互独立的、作用差不多大的随机因素影响下形成的随机变量，其极限分布为正态分布。

- **χ^2 分布**

若 $X_1, X_2, \cdots, X_n$ 为相互独立的、服从标准正态分布 $N(0,1)$ 的随机变量，则它们的平方和 $Y = \sum_{i=1}^{n} X_i^2$ 服从 χ^2 分布，记作 $Y \sim \chi^2(n)$，n 称自由度，它的期望 $EY = n$，方差 $DY = 2n$。

- **t 分布**

若 $X \sim N(0,1)$，$Y \sim \chi^2(n)$，且相互独立，则 $T = \frac{X}{\sqrt{Y/n}}$ 服从 t 分布，记作 $T \sim t(n)$，n 称自由度。t 分布又称**学生氏 (Student) 分布**。

t 分布的密度函数曲线和 $N(0,1)$ 曲线形状相似。理论上 $n \to \infty$ 时，$T \sim t(n) \to N(0,1)$，实际上当 $n > 30$ 时它与 $N(0,1)$ 就相差无几了。

- **F 分布**

若 $X \sim \chi^2(n_1), Y \sim \chi^2(n_2)$, 且相互独立，则 $F = \frac{X/n_1}{Y/n_2}$ 服从 F 分布，记作 $F \sim F(n_1, n_2)$，(n_1, n_2) 称自由度。

2.3 正态总体统计量的分布

用样本来推断总体，需要知道样本统计量的分布，而样本又是一组与总体同分布的随机变量，所以样本统计量的分布依赖于总体的分布。当总体服从一般的分布时，求某个样本统计量的分布是很困难的，只有在总体服从正态分布时，一些重要的样本统计量 (均值、标准差) 的分布才有便于使用的结果。另一方面，现实生活中需要进行统计推断的总体，多数可以认为服从 (或近似服从) 正态分布，所以统计中人们在正态总体的假定下研究统计量的分布，是必要的与合理的。

设总体 $X \sim N(\mu, \sigma^2)$ $x_1, x_2, \cdots, x_n$ 为一容量 n 的样本，其均值 $\bar{x}$ 和标准差 s 由公式确定，则用 $\bar{x}$ 和 s 构造的下面几个分布在统计中是非常有用的。

$$\bar{x} \sim N(\mu, \frac{\sigma^2}{n}) 或 \frac{\bar{x} - \mu}{\sigma/\sqrt{n}} \sim N(0,1)$$

$$\frac{(n-1)s^2}{\sigma^2} \sim \chi^2(n-1)$$

$$\frac{\bar{x} - \mu}{s/\sqrt{n}} \sim t(n-1)$$

设有两个总体 $X \sim N(\mu_1, \sigma^2)$ 和 $Y \sim N(\mu_2, \sigma^2)$，及由容量分别为 n，n 的两个样本确定的均值 $\bar{x}, \bar{y}$ 和标准差 s_1, s_2，则

$$\frac{(\bar{x} - \bar{y}) - (\mu_1 - \mu_2)}{\sqrt{\sigma_1^2/n_1 + \sigma_2^2/n}} \sim N(0,1)$$

$$\frac{(\bar{x} - \bar{y}) - (\mu_1 - \mu_2)}{s_w\sqrt{1/n_1 + 1/n_2}} \sim t\,(n_1 + n_2 - 2)$$

其中，$s_w^2 = \frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{n_1+n_2-2}$，

$$\frac{s_1^2/\sigma_1^2}{s_2^2/\sigma_2^2} \sim F(n_1 - 1, n_2 - 1)$$

对于上式，假定 $\sigma_1 = \sigma_2$，但它们未知，于是用 s 代替。在下面的统计推断中我们要反复用到这些分布。

2.4 参数估计

利用样本对总体进行统计推断的一类问题是参数估计，即假定已知总体的分布，通常是 $X \sim N(\mu, \sigma^2)$，估计有关的参数，如 μ, σ^2 。参数估计分点估计和区间估计两种。

- **点估计**

点估计是用样本统计量确定总体参数的一个数值。评价估计优劣的标准有无偏性、最小方差性、有效性等，估计的方法有矩法、极大似然法等。

最常用的是对总体均值 μ 和方差 σ^2 (或标准差 σ) 作点估计。让我们暂时抛开评价标准，当从一个样本算出样本均值 $\bar{x}$ 和方差 s^2 后，对 μ 和 σ^2(或 σ) 一个自然、合理的点估计显然是 (在字母上加表^ 示它的估计值)

$$\hat{\mu} = \bar{x}, \hat{\sigma}^2 = s^2, \hat{\sigma} = s$$

- **区间估计**

点估计虽然给出了待估参数的一个数值，却没有告诉我们这个估计值的精度和可信程度。一般地，总体的待估参数记作 θ(如 μ, σ^2)，由样本算出的 θ 的估计量记作 $\hat{\theta}$ ，人们常希望给出一个区间 $[\hat{\theta}_1, \hat{\theta}_2]$，使 θ 以一定的概率落在此区间内。若有

$$P\{\hat{\theta}_1 < \theta < \hat{\theta}_2\} = 1 - \alpha, 0 < \alpha < 1$$

则 $[\hat{\theta}_1, \hat{\theta}_2]$ 称为 θ 的置信区间，$\hat{\theta}_1, \hat{\theta}_2$ 分别称为置信下限和置信上限，$1-\alpha$ 称为置信概率或置信水平，α 称为显著性水平。

给出的置信水平为 $1-\alpha$ 的置信区间 $[\hat{\theta}_1, \hat{\theta}_2]$，称为 θ 的区间估计。置信区间越小，估计的精度越高; 置信水平越大，估计的可信程度越高。但是这两个指标显然是矛盾的，通常是在一定的置信水平下使置信区间尽量小。通俗地说，区间估计给出了点估计的误差范围。

2.5 假设检验

统计推断的另一类重要问题是假设检验问题。在总体的分布函数完全未知或只知其形式但不知其参数的情况，为了推断总体的某些性质，提出某些关于总体的假设。例如，提出总体服从泊松分布的假设，又如对于正态总体提出数学期望等于 μ_0 的假设等。假设检验就是根据样本对所提出的假设做出判断: 是接受还是拒绝。这就是所谓的假设检验问题。

2.5.1 单个总体 $N(\mu, \sigma^2)$ 均值 μ 的检验

假设检验有三种:
双边检验:$H_0: \mu = \mu_0$，$H_1: \mu \neq \mu_0$;
右边检验:$H_0: \mu \leqslant \mu_0$，$H_1: \mu > \mu_0$;
左边检验:$H_0: \mu \geqslant \mu_0$，$H_1: \mu < \mu_0$;

- σ^2 已知，关于 μ 的检验 (**Z** 检验)

在 Matlab 中 **Z** 检验法由函数 ztest 来实现，命令为

$$[h, p, ci] = ztest(x, mu, sigma, alpha, tail)$$

其中输入参数 x 是样本，mu 是 H_0 中的 μ_0 ，sigma 是总体标准差 σ ，alpha 是显著性水平 α(alpha 缺省时设定为 0.05)，tail 是对备选假设 H_1 的选择: H_1 为 $\mu \neq \mu_0$ 时用 tail=0(可缺省); H_1 为 $\mu > \mu_0$ 时用 tail=1; H_1 为 $\mu < \mu_0$ 时用 tail=-1。输出参数 h=0 表示接受 H_0，h=1 表示拒绝 H_0 ，p 表示在假设 H_0 下样本均值出现的概率，p 越小 H_0 越值得怀疑，ci 是 μ_0 的置信区间。

- σ^2 未知，关于 μ 的检验 (t 检验)

  在 Matlab 中 t 检验法由函数 ttest 来实现，命令为

$$[h, p, ci] = ttest(x, mu, alpha, tail)$$

2.5.2 分布拟合检验

在实际问题中，有时不能预知总体服从什么类型的分布，这时就需要根据样本来检验关于分布的假设。下面介绍 χ^2 检验法和专用于检验分布是否为正态的“偏峰、峰度检验法”。

- χ^2 **检验法**

什么是 χ^2 检验? χ^2 检验就是在总体的分布未知的情况下, 根据样本 $X_1, X_2, \cdots, X_n$ 来检验关于总体分布的假设。一般情况下，我们会设置原假设：H_0：总体 X 的分布函数为 F(x), 进而得到备择假设 H_1：总体 X 的分布函数不是 F(x), 然后运用统计学方法进行检验。值得注意的是，若总体 X 为离散型，则上述假设写为 H_0：总体 X 的分布律为 $P\{X = t\} = p, i = 1, 2, \cdots$. 若总体 X 为连续型，则上述假设写为 H_0：总体 X 的概率密度为 $f(x)$. 此外，在使用 χ^2 检验法检验假设 H_0 时, 若 $F(x)$ 的形式已知, 但有参数值未知, 需要先用最大似然估计法估计参数, 然后作检验.

χ^2 检验的整体思路，就是局部样本的某些统计学特征和整体样本相似。将随机试验可能结果的全体 Ω 分为 m 个互不相容的事件 $A_1, A_2, \cdots, A_m (\sum_{i=1}^{m} A_i = \Omega, A_i A_j = \Phi, i \neq j, i, j = 1, 2, \cdots, m)$. 于是在假设 H_0 下, 我们可以计算 $p_{i0} = P(A_i), i = 1, 2, \cdots, m$. 在 n 次试验中, 事件 A_i 出现的频率 $\frac{N_i}{n}$ 与 p_{i0} 往往有差异, 但一般来说, 若 H_0 为真, 且试验次数又多时, 这种差异不应很大.

2.5.3 连续分布的检验

正太性检验：定性方法: 正态 QQ 图 qqnorm( )

定量方法: 国际标准 ISO

夏皮洛-威尔克检验 Shapiro-Wilk (W 检验)

爱泼斯-普利检验 Epps-Pully

其他的一些分布：柯尔莫哥洛夫检验，检验分布是否服从某种理论分布；斯米尔诺夫检验，检验两个样本是否服从同一分布。

2.6 数据预处理

数据清洗要成什么样子?

下面我们以 Python 中的 NumPy、Pandas、Matplotlib 为例进行数据清洗等预处理。

我们平时拿到的数据一般来说都是杂乱无章的，对数据清洗的首要步骤就是明确我们手上的数据究竟是什么样子的，而预处理的步骤也十分简单。

Country N	Country C	Series Na	Series Co	1960 [YR1	1961 [YR1	1962 [YR1	1963 [YR1	1964 [YR1	119
Afghanist	AFG	CO2 emiss	EN.ATM.CO	0.04606	0.053604	0.073765	0.074233	0.086292	0.
Afghanist	AFG	GDP per c	NY.GDP.PC	..	..	..	..	..	..
Afghanist	AFG	Populatio	SP.POP.TO	8996351	9166764	9345868	9533954	9731361	9
Afghanist	AFG	Populatio	EN.POP.DN	..	14.04093	14.31527	14.60337	14.90574	15
Afghanist	AFG	Urban pop	SP.URB.TO	8.401	8.684	8.976	9.276	9.586	
Afghanist	AFG	Industry	NV.IND.TO	..	..	..	..	..	..
Albania	ALB	CO2 emiss	EN.ATM.CO	1.258195	1.374186	1.439956	1.181681	1.111742	1.
Albania	ALB	GDP per c	NY.GDP.PC	..	..	..	..	..	..
Albania	ALB	Populatio	SP.POP.TO	1608800	1659800	1711319	1762621	1814135	1
Albania	ALB	Populatio	EN.POP.DN	..	60.57664	62.4569	64.32923	66.20931	68
Albania	ALB	Urban pop	SP.URB.TO	30.705	30.943	31.015	31.086	31.158	
Albania	ALB	Industry	NV.IND.TO	..	..	..	..	..	..
Algeria	DZA	CO2 emiss	EN.ATM.CO	0.553764	0.53181	0.484954	0.452824	0.459569	0.
Algeria	DZA	GDP per c	NY.GDP.PC	2466.038	2078.222	1628.389	2133.319	2200.814	22
Algeria	DZA	Populatio	SP.POP.TO	11124888	11404859	11690153	11985136	12295970	12
Algeria	DZA	Populatio	EN.POP.DN	..	4.788457	4.908241	5.032093	5.1626	5.
Algeria	DZA	Urban pop	SP.URB.TO	30.51	31.797	33.214	34.662	36.141	
Algeria	DZA	Industry	NV.IND.TO	..	..	..	..	..	..

图 1: 未处理的数据

```python
import numpy as np
import pandas as pd

#导入、预览数据
df = pd.read_csv('train.csv')

#检查数据情况
df.head(3)
df.info()

#某个特征值的数量情况
df['Sex'].value_counts() #性别变量的种类及数量情况

#缺失值处理:
df.isnull().sum() #检查空缺值，并加以统计
df[df['Age'] == np.nan] = 0 #令某个索引下的空值为 0
df.fillna(0).head(3) #填 0 操作
df.dropna().head(3) #删除空值

#重复值处理:
df[df.duplicated()] #检查重复值
df.drop_duplicates().head() #删除重复值
df.to_csv('test_clear.csv') #保存 csv 文件

#分组操作:
#第一种方式
df['AgeBand'] = pd.cut(df['Age'], 5,labels = ['1','2','3','4','5'])

#第二种方式
df['AgeBand'] = pd.cut(df['Age'],[0,5,15,30,50,80],labels = ['1','2','3','4','5'])

#第三种方式
df['AgeBand'] = pd.qcut(df['Age'],[0,0.1,0.3,0.5,0.7,0.9],labels = ['1','2','3','4','5'])

#键值改名:
#方法一
df['Sex_num'] = df['Sex'].replace(['male','female'],[1,2])
```

countryname	year	CO2	GDP	POP_D	POP_T	INDUS	URBAN
Albania	1995	0.65453713	1703.237	116.34248	3187784	22.79632	38.911
Albania	1996	0.63662531	1869.817	115.62164	3168033	17.37384	39.473
Albania	1997	0.49036506	1676.083	114.90077	3148281	18.10977	40.035
Albania	1998	0.56027144	1835.597	114.17993	3128530	15.34684	40.601
Albania	1999	0.96016441	2085.371	113.45905	3108778	16.53674	41.169
Albania	2000	0.97817468	2244.565	112.73821	3089027	19.39569	41.741
Albania	2001	1.0533042	2453.558	111.68515	3060173	21.73577	42.435
Albania	2002	1.2295407	2572.652	111.35073	3051010	23.10637	43.501
Albania	2003	1.4126972	2725.097	110.93489	3039616	24.05561	44.573
Albania	2004	1.3762127	2887.291	110.47223	3026939	24.66556	45.651
Albania	2005	1.4124982	3062.593	109.90828	3011487	25.08161	46.731
Albania	2006	1.3025764	3263.812	109.21704	2992547	25.30803	47.815
Albania	2007	1.3223349	3485.228	108.39478	2970017	25.34033	48.902
Albania	2008	1.4843111	3775.48	107.5662	2947314	25.16101	49.991
Albania	2009	1.4956002	3928.342	106.84376	2927519	24.41378	51.076
Albania	2010	1.5785736	4094.36	106.31464	2913021	24.93746	52.163
Albania	2011	1.8037147	4210.077	106.02901	2905195	24.48448	53.247
Albania	2012	1.6929083	4276.918	105.85405	2900401	22.91855	54.33
Albania	2013	1.7492111	4327.608	105.66029	2895092	23.06132	55.387
Albania	2014	1.9787633	4413.335	105.44175	2889104	21.50885	56.423
Angola	1995	0.76917343	1878.793	11.445411	14268994	60.28518	44.169
Angola	1996	0.71230634	2073.215	11.776918	14682284	67.82068	45.346
Angola	1997	0.48920938	2164.082	12.103137	15088981	60.75774	46.525
Angola	1998	0.47137391	2204.91	12.436286	15504318	55.15729	47.71

图 2: 处理后的面板数据

```
#方法二
df['Sex_num'] = df['Sex'].map({'male': 1, 'female': 2})
```

表 1: 功能函数概览

函数名	说明
count	分组中非 NA 值的数量
sum	非 NA 值的和
mean	非 NA 值的平均值
median	非 NA 值的算术中位数
std、var	无偏（分母为 n-1）标准差和方差
min、max	非 NA 值的最小值和最大值
prod	非 NA 值的积
first、last	第一个和最后一个非 NA 值

2.7 数据重构

数据重构就是整理多张表格，包括但不限于：合并、拆分，并利用分组初步找到数据的基础统计信息。

```
#合并两张表 table_1 与 table_2
table_1 = pd.read_csv("data/train-left-up.csv")
table_2 = pd.read_csv("data/train-right-up.csv")

#方法一:不是很推荐
connect_table = [table_1,table_2]
```

```
result_1 = pd.concat(connect_table,axis=1) #纵向合并为axis=0

#方法二:功能强大
result_1 = table_1.join(table_2)
result_2 = table_3.join(table_4)
result = result_1.append(result_2)

#方法三:方法二的拓展
result_1 = pd.merge(table_1,table_2,left_index=True,right_index=True)
result_2 = pd.merge(table_3,table_4,left_index=True,right_index=True)
result = resul_1.append(result_2)

#保存文件
result.to_csv('result.csv')

#分组处理
#任务一:计算男女分别平均票价
df = text['Fare'].groupby(text['Sex'])
means = df.mean()
means

#任务二:计算男女分别存活人数
survived_sex = text['Survived'].groupby(text['Sex']).sum() survived_sex.head()

#任务三:计算不同舱位存活人数
survived_pclass = text['Survived'].groupby(text['Pclass']) survived_pclass.sum()

#任务四:综合不同舱位、不同年龄的票价平均值
text.groupby(['Pclass','Age'])['Fare'].mean().head()

#合并任务一、任务二并保存
result = pd.merge(means,survived_sex,on='Sex') result
result.to_csv('sex_fare_survived.csv')

#功能函数应用示例:
#找出最大值的年龄段
survived_age[survived_age.values==survived_age.max()]

#算存活率,或者其他的统计信息
#首先计算总人数
_sum = text['Survived'].sum()
print("sum of person:"+str(_sum))
precetn =survived_age.max()/_sum
print("最大存活率:"+str(precetn))
```

2.8 数据可视化

2.8.1 复合可视化图

直方-曲线可视复合图

```
#直方-曲线可视复合图
import seaborn as sns
from matplotlib import pyplot as plt

```

```
comp1 = np.random.normal(0, 1, size=200)
comp2 = np.random.normal(10, 2, size=200)
values = pd.Series(np.concatenate([comp1, comp2]))
figure_demo1=sns.distplot(values, bins=100, color='k')
demo2=figure_demo1.get_figure()
demo2.savefig('./demo2', dpi = 400)
```

散点-归回线可视复合图

```
#散点-归回线可视复合图
import seaborn as sns
from matplotlib import pyplot as plt

macro = pd.read_csv('examples/macrodata.csv')
data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]
trans_data = np.log(data).diff().dropna()
trans_data[-5:]
plt.figure()
sns.regplot('m1', 'unemp', data=trans_data)
plt.title('Changes in log %s versus log %s' % ('m1', 'unemp'))
plt.savefig('./demo3', dpi = 400)
```

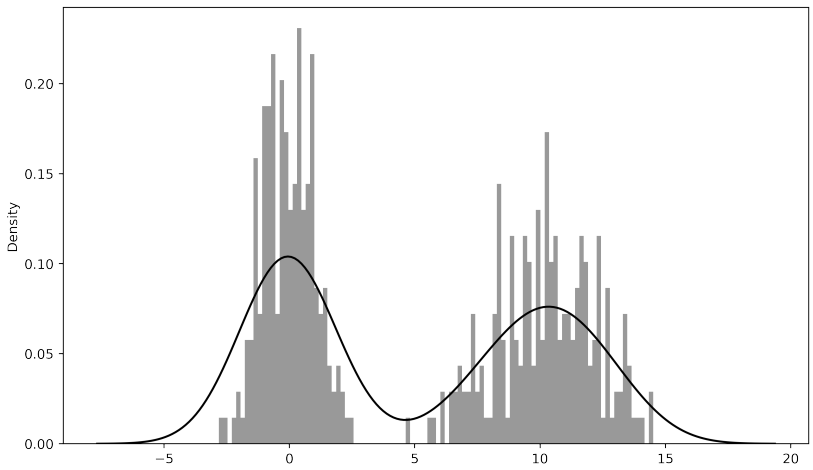


图 3: 直方-曲线复合图

2.8.2 折线图

```
#折线图
import seaborn as sns
from matplotlib import pyplot as plt

#阶梯状插值
plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')

# 风格可选:
# 'Solarize_Light2', '_classic_test_patch', 'bmh'
```

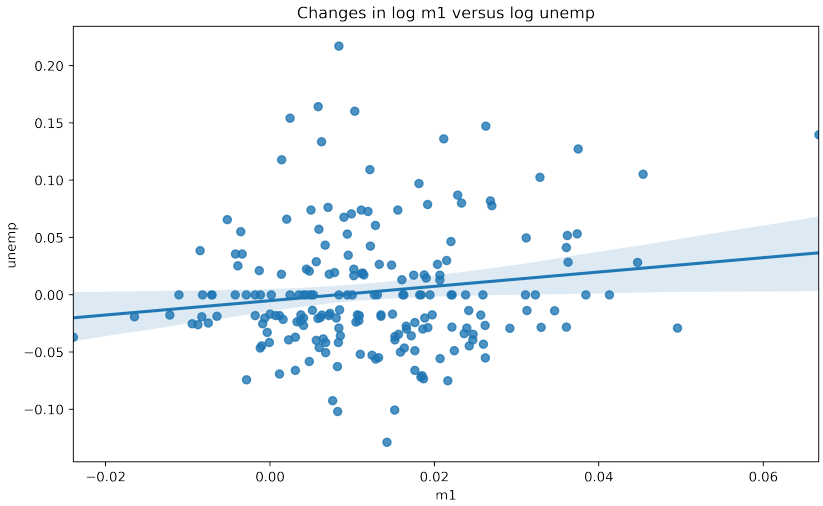


图 4: 散点-回归复合图

```
# 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale'
# 'seaborn', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette'
# 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook'
# 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk',
# 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind10'

ages_x = [25,26,27,28,29,30,31,32,33,34,35] #设置 x 轴的取值
py_dev_y = [45372,48876,53850,57287,63016,65998,70003,70000,71496,75370,83640]
plt.plot(ages_x, py_dev_y, color='#5a7d9a', linewidth=3, label='Python')

js_dev_y = [37810,43515,46823,49293,53437,56373,62375,66674,68745,68746,74583]
plt.plot(ages_x, js_dev_y, color='#adad3b', linewidth=3, label='JavaScript')

dev_y = [38496,42000,46752,49320,53200,56000,62316,64928,67317,68748,73752]
plt.plot(ages_x, dev_y, color='#444444', linestyle='--', label='All Devs')

plt.xlabel('Ages')
plt.ylabel('Median Salary (USD)')
plt.title('Median Salary (USD) by Age')

plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

2.8.3 折线注释图

```
#折线注释图
from datetime import datetime

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
data = pd.read_csv('examples/spx.csv', index_col=0, parse_dates=True)
```

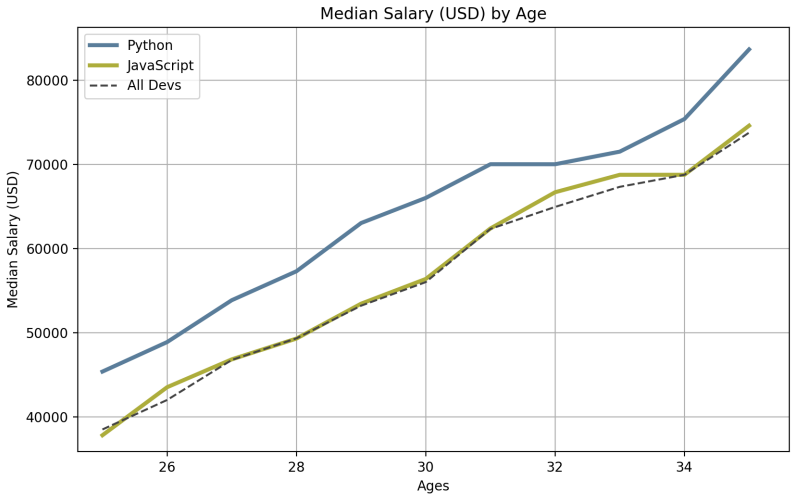


图 5: 优化后的折线图

```
spx = data['SPX']
spx.plot(ax=ax, style='k-')

#图片中注释部分
crisis_data = [
(datetime(2007, 10, 11), 'Peak of bull market'),
(datetime(2008, 3, 12), 'Bear Stearns Fails'),
(datetime(2008, 9, 15), 'Lehman Bankruptcy')]

arrowprops=dict(facecolor='black', headwidth=4, headlength=4),
horizontalalignment='left', verticalalignment='top')

# Zoom in on 2007-2010
ax.set_xlim(['1/1/2007', '1/1/2011'])
ax.set_ylim([600, 1800])
ax.set_title('Important dates in the 2008-2009 financial crisis' )
plt.savefig('./demo4', dpi = 400)
```

2.8.4 Pie 图

```
#Pie图
plt.style.use("fivethirtyeight")
slices = [59219,55466,47544,36443,35917]
our_label = ['JavaScript', 'HTML/CSS', 'SQL', 'Python','Java']
our_explode = [0,0,0,0.1,0]
plt.pie(slices, labels=our_label, explode=our_explode, shadow=True,
            startangle=90, autopct='%1.1f%%',
            wedgeprops={'edgecolor': 'black'})
plt.title("My Awesome Pie Chart")
plt.tight_layout()
plt.show()
```

2.8.5 Stack 图

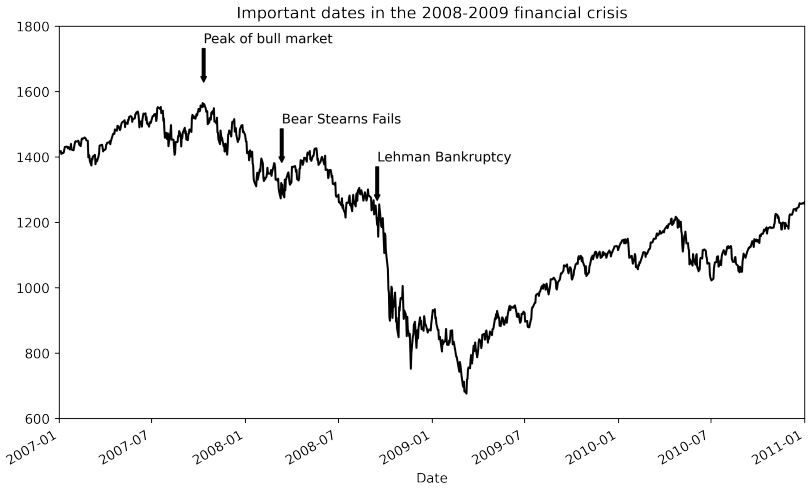


图 6: 折线注释图

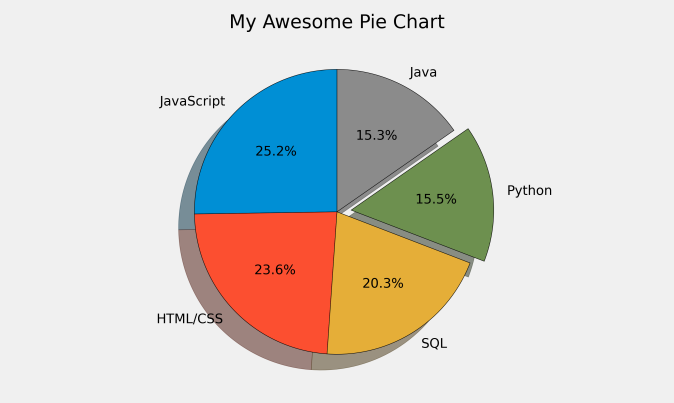


图 7: Pie 图

```
#Stack图
plt.style.use("fivethirtyeight")
minutes = [1,2,3,4,5,6,7,8,9]
player1 = [1,2,3,3,4,4,4,4,5]
player2 = [1,1,1,1,2,2,2,3,4]
player3 = [1,1,1,2,2,2,3,3,3]
our_labels = ['player1','player2','player3']
plt.stackplot(minutes, player1, player2, player3,labels= our_labels)
plt.legend(loc='upper left')
plt.title("My Awesome Stack Plot")
plt.tight_layout()
```

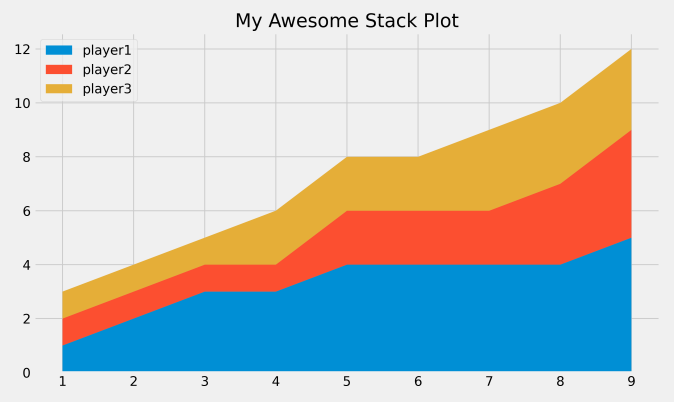

My Awesome Stack Plot
player1
player2
player3
0
2
4
6
8
10
12
1
2
3
4
5
6
7
8
9

图 8: Stack 图

3 OLS 回归分析

- **假设 1:** 二者之间存在线性关系 (解释变量 X 是非随机的，被解释变量 y 是随机的)

$$y = a_0 + a_1 * x_1 + a_2 * x_2 + \cdots + a_k * x_k + u \tag{1}$$

$$y = Xb + u \tag{2}$$

- **假设 2:** X 是满秩的，i.e. $rank(X) = k$. 满秩

- **假设 3:** 干扰项的条件期望为零 (模型的设定是正确的)，

即:

$$E[Xu] = 0 \quad \text{or} \quad E[u|X] = 0 \tag{3}$$

也就是说：我们假定干扰项和解释变量 X 之间没有相关性，在 X 条件下，干扰项是完全随机的，其均值为 0，或者说两者的交集的均值为 0. 事实上这种假设很强，我们在后面会说明。

我们如何表示干扰项 u 呢？通过（2）式可得：$u = y–Xb$，用 X^T 表示 X 的转置矩阵，那么（3）式可以表达为：$E[X^T(y - Xb)] = 0$，展开运算可得：$b = (X^TX)^{-1} * X^Ty$，因此有:

$$\begin{aligned} E[b] &= E[(X^TX)^{-1}X^T * (Xb_0 + u)] \\ &= b_0 + (X^TX)^{-1}X^TE[u] \\ &= b_0 \end{aligned}$$

上面的含义即为：系数 b 的期望值我们记作 b_0，OLS 估计量是真实值的无偏估计。于是可以得到 y 的预测值 $\hat{y} = X * b$，残差 $e = y - \hat{y} = y - Xb$.

模型已经建立完成，那么我们所建的模型质量如何呢? 在这里面我们的 b 是随机的，因为 y 就是随机的，我们的目的就是求 y 的预测值，所以 b 可能出现任何值。我们都知道，假设 3 的约束性太松散，干扰项有很多，但是干扰程度可能不同。于是我们加紧约束一点:

- **假设 4:（同方差假设）** $Var[u_i|X] = \sigma^2$ 我们假设在 X 的条件下，所有干扰项的方差都是同一个值，因此称为同方差假设。这种假设约束性很强，结合假设 3 我们可知：$u_i \sim (0, \sigma^2)$，也就是说干扰项服从正态分布

什么是标准误差（s.e.，即：均方根误差)? 假设 n 个测量值的误差为 $E_1, E_2, \cdots, E_n$，那么这组测量值的标准差为：$\sigma = \sqrt{\frac{E_1^2+E_2^2+\cdots+E_n^2}{n}} = \sqrt{\frac{\sum E_i^2}{n}}$

但这样有个问题：由于被测量的真值是未知的，各个测量值的误差我们没办法得到，因此不能按照上式求得。真实值大概是多少呢? 我们一般认为：该组数据的算术平均值是最接近真实值的，因此容易算出测量值和该算术平均值的差，记为残差。例如:

$$11, \quad 22, \quad 35, \quad 44, \quad 64, \quad 87$$

这六个数据的方差为 780.57，平均值为 43.83，标准误为 11.4。如何直观理解这些数据呢? 事实上，该组数据平均每个数据的方差为 $780.57/6 = 130.095$，开方即可得到标准误为 11.4，可以看作是：若我们把期望值作为真实值，标准误差表示每个值和真实值平均误差。

需要注意的是，标准误差不是测量值的实际误差，也不是误差范围，它只是对一组数据可靠性

的估计。标准误差小，测量的可靠性大，反之测量就不大可靠。在上面的数据中，并不说每个数据真实值在上下 11.4 浮动，而是说相比该组数据，11.4 的数值还算是可靠的（当然是在平均值作为真实值参考的前提下）。

因为干扰项 u_i 服从正态分布，$u_i \sim i.i.dN(0, \sigma^2)$，因为 b 是 y 的线性组合，于是我们可以得到：$b \sim N(b_0, Var(b))$，即：$(b_j - b_{j0})/s.e.(b_j) \sim t(n-k)$

通常我们得到：每一个变量系数和真实值之间的差，除以标注误符合 t 分布，n 是观测样本数，k 是解释变量的个数。如何检验呢？我们假设 H0：$b_{j0} = 0$，观察真实值是否显著异零，如果不显著异于零，那么证明这个解释变量对因变量没有什么影响，如果这个值对真实值没有影响，我们可以得到 $b_j/s.e.(b_j) \sim t(n-k)$，从而对结果进行检验分析。t 值的计算：t= 系数/标准误，由于 t 值服从 t 分布，所以我们很容易计算其 p 值。

t 检验可用于比较男女身高是否存在差别。举个例子，为了进行独立样本 t 检验，需要一个自（分组）变量（如性别：男、女）与一个因变量（如身高测量值）。根据自变量的特定值，比较各组中因变量的均值。用 t 检验比较下列男、女儿童身高的均值。

- 假设：H_0：男平均身高 = 女平均身高；
  H_1：男平均身高 ≠ 女平均身高。选用双侧检验：选用 α=0.05 的统计显著水平
- 从输出结果查看 t 检验的 p 值，是否达到显著水平：是，接受 H1，男平均身高与女平均身高不同；否，接受 H0，尚无证据支持男女身高差异。

表 2: 男女生身高数据

被测试者	性别	身高（cm）
对象 1	男性	111
对象 2	男性	110
对象 3	男性	109
对象 4	女性	102
对象 5	女性	104

检验系数显著性的新方法: **自体抽样法 (Bootstrap)**：假设样本是母体中随机抽取的; 通过反复从样本中抽取样本来模拟母体的分布;

1. 从样本中可重复地抽取 N 个样本，执行 OLS，记录系数估计值;
2. 将第一步重复进行 300 次，得到系数估计值的 300 个记录;
3. 统计这 300 个估计值的标准误 se_bs，将其视为实际估计值的标准误;
4. 计算 t 值和相应的 p 值

方差分析，y 的总波动 = 模型能够解释的波动 + 残差的波动，我们关心的肯定是模型能解释的方差波动占比多少，有没有专门的统计量来描述它呢？有的，这就是 R^2，也就是模型能够解释的波动占总波动的比例。

4 GLS 广义最小平方法

4.1 传统回归模型

对于如下回归模型：

$$y_i = \alpha_0 + \alpha_1 x_{i1} + \cdots + \alpha_k x_{ik} + \varepsilon_i$$

在 OLS 回归分析部分，我们假设：

$$Var(\varepsilon_i) = \sigma^2$$

换言之，干扰项的方差-协方差矩阵为：

$$\mathrm{Var}(\varepsilon) = \mathbf{\Sigma} = \begin{bmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & \sigma^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma^2 \end{bmatrix}$$

然而，这个假设往往过于严格，并不能很好地描述现实中的经济问题。

4.2 放松同方差假设

$$\mathrm{Var}(\varepsilon) = \sigma^2\mathbf{\Sigma} = \sigma^2 \begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2 & \cdots & \sigma_1\sigma_n \\ \sigma_2\sigma_1 & \sigma_2^2 & \cdots & \sigma_2\sigma_n \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_n\sigma_1 & \sigma_n\sigma_2 & \cdots & \sigma_n^2 \end{bmatrix}$$

4.3 GLS 基本思想

考虑如下模型，

$$y_i = \alpha_0 + \alpha_1 x_i + \alpha_2 w_i + \mu_i, \quad i = 1, 2, \ldots, n$$

其中，$Var(\mu_i) = \sigma_i^2 = \sigma^2 x_i^2$，即

$$\mathrm{Var}(\boldsymbol{\mu}) = \sigma^2\mathbf{\Sigma} = \sigma^2 \begin{bmatrix} x_1^2 & 0 & \cdots & 0 \\ 0 & x_2^2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & x_n^2 \end{bmatrix}$$

事实上，从 $Var(\mu_i) = \sigma^2 x_i^2$ 中我们可以看出，只要在原模型 $y_i = \alpha_0 + \alpha_1 x_i + \alpha_2 w_i + \mu_i$ 两边同时除以 x_i，即可得到干扰项具有同方差的模型：

$$y_i/x_i = \alpha_0/x_i + \alpha_1 + \alpha_2 w_i/x_i + \mu_i/x_i$$

$$y_i^* = \alpha_0 + \alpha_1 x_i^* + \alpha_2 w_i^* + \mu_i^*$$

其中，$y_i^* = y_i/x_i, \mu_i^* = \mu_i/x_i$。显然，

$$\mathrm{Var}(\mu_i/x_i) = \sigma^2$$

所以，转换后面的模型是一个传统的线性回归模型，其 OLS 估计是 BLUE 的。

由此可见，我们只需要对转换后的模型执行 OLS 估计即可得到原始模型的无偏一致估计量。在上面的模型中，

$$\Sigma^{-1} = G^T G$$

于是，我们可以得到:

$$\mathbf{y}^* = \mathbf{G}\mathbf{y} = \begin{bmatrix} 1/x_1 & 0 & \cdots & 0 \\ 0 & 1/x_2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & 1/x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} y_1/x_1 \\ y_2/x_2 \\ \vdots \\ y_n/x_n \end{bmatrix}$$

$$\mathbf{X}^* = \mathbf{G}\mathbf{X} = \begin{bmatrix} 1/x_1 & 0 & \cdots & 0 \\ 0 & 1/x_2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & 1/x_n \end{bmatrix} \begin{bmatrix} 1 & x_1 & w_1 \\ 1 & x_2 & w_2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & w_n \end{bmatrix} = \begin{bmatrix} 1/x_1 & 1 & w_1/x_1 \\ 1/x_2 & 1 & w_2/x_2 \\ \vdots & \vdots & \vdots \\ 1/x_n & 1 & w_n/x_n \end{bmatrix}$$

4.4 GLS 矩阵表述

4.4.1 GLS 估计：Σ 已知

模型:

$$y = X\beta + \varepsilon$$

其中，

$$Var(\varepsilon) = \sigma^2\Sigma$$

由于方差-协方差矩阵 Σ 是正定的，所以所以它的逆矩阵也是正定的，于是存在矩阵 G 满足:

$$\mathbf{\Sigma}^{-1} = \mathbf{G}'\mathbf{G} \quad \text{和} \quad \mathbf{G}\mathbf{\Sigma}\mathbf{G}' = \mathbf{I}_n$$

两边同乘 G，得到:

$$\mathbf{G}\mathbf{y} = \mathbf{G}\mathbf{X}\boldsymbol{\beta} + \mathbf{G}\varepsilon$$

可进一步表示为:

$$y^* = X^*\beta + \varepsilon^*$$

其中，$\mathbf{y}^* = \mathbf{G}\mathbf{y}, \mathbf{X}^* = \mathbf{G}\mathbf{X}, \quad \boldsymbol{\varepsilon}^* = \mathbf{G}\varepsilon$，变换后的随机干扰项具有均齐方差:

$$Var(\varepsilon^*) = GVar(\varepsilon)G^T = \sigma^2 I_n$$

模型满足经典回归模型的所有假设，因此可以采用 OLS 进行估计，而前面介绍的所有针对经典回归模型的统计推断方法也同样适用于该模型。OLS 估计称为原模型的 GLS 估计量:

$$\begin{aligned}\hat{\boldsymbol{\beta}} &= \left(\mathbf{X}^{*\prime}\mathbf{X}^{*}\right)^{-1}\mathbf{X}^{*\prime}\mathbf{y}^{*} \\ &= \left(\mathbf{X}'\mathbf{G}'\mathbf{G}\mathbf{X}\right)^{-1}\mathbf{X}'\mathbf{G}'\mathbf{G}\mathbf{y} \\ &= \left(\mathbf{X}'\boldsymbol{\Sigma}^{-1}\mathbf{X}\right)^{-1}\mathbf{X}'\ {}^{-1}\mathbf{y}\end{aligned}$$

相应的方差矩阵为:

$$\mathrm{Var}(\hat{\boldsymbol{\beta}}) = \sigma^2\left(\mathbf{X}^{*\prime}\mathbf{X}^{*}\right)^{-1} = \sigma^2\left(\mathbf{X}'\mathbf{G}'\mathbf{G}\mathbf{X}\right)^{-1} = \sigma^2\left(\mathbf{X}\boldsymbol{\Sigma}^{-1}\mathbf{X}\right)^{-1}$$

显然，σ^2 的一个无偏估计量为:

$$s^2 = \frac{\left(\mathbf{y}^{*} - \mathbf{X}^{*}\hat{\boldsymbol{\beta}}\right)'\left(\mathbf{y}^{*} - \mathbf{X}^{*}\hat{\boldsymbol{\beta}}\right)}{N-K}$$

4.4.2 FGLS 估计：Σ 未知

前面讲到的 GLS 估计方法是以 Σ 已知为前提假设的，但不幸的是，多数情况下 Σ 中往往包含着众多的未知参数。因此，我们需要采用以下两步法进行估计:

1. 估计 Σ 中的未知参数，得到 $\hat{\Sigma}$;
2. 利用第一步得到的 $\hat{\Sigma}$ 进行 GLS 估计，得到:

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}'^{\wedge-1}\mathbf{X}\right)^{-1}\mathbf{X}'^{\wedge-1}\mathbf{y} \quad \text{和} \quad \mathrm{Var}(\hat{\boldsymbol{\beta}}) = \sigma^2\left(\mathbf{X}'^{\wedge-1}\mathbf{X}\right)^{-1}$$

在实际应用过程中，我们往往将上述” 两步法” 重复进行多次，直到相邻两次的估计结果差别很小为止 (称为” 收敛”)。这种做法有助于提高估计的有效性。

4.5 Σ 的设定

4.5.1 截面数据: 异方差

$$\mathrm{Var}(\varepsilon) = \sigma^2\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n^2 \end{bmatrix}$$

4.5.2 时间序列: 自相关

$$\mathrm{Var}(\boldsymbol{\varepsilon}) = \sigma^2\boldsymbol{\Sigma} = \sigma^2\begin{bmatrix} 1 & \rho & \rho^2 & \cdots & \rho^{T-1} \\ \rho & 1 & \rho & \cdots & \rho^{T-2} \\ \rho^2 & \rho_1 & 1 & \cdots & \rho^{T-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho^{T-1} & \rho^{T-2} & \rho^{T-3} & \cdots & 1 \end{bmatrix}$$

5 时间序列分析

5.1 简单移动平均法

如果数据具有一定的规律性，且一直在某一个值附近波动，那么我们可以用该组数据的平均值来近似拟合下一次可能出现的值。例如：

$$2, \quad 3, \quad 3, \quad 4, \quad 2, \quad 3, \quad 3, \quad 5, \quad 2, \quad 3$$

我们一共有 10 个数据，且每一个数据大概都在 3 附近波动。简单移动平均法告诉我们，能够预测的项数 t 一定小于 10，这是十分显然的。假设我们的任务是预测未来的五项值是多少，那么 y_{11} 是多少？我们用它之前十次数据取平均值来确定，即：$y_{11} = \frac{1}{10}(y_{10}+\cdots+y_1)$，同理 $y_{12} = \frac{1}{10}(y_{11}+\cdots+y_2)$，其预测标准误差为相邻两个值的差平方和与 N 期预测数之比：

$$S = \sqrt{\frac{\sum\limits_{t=N+1}^{T}(y_t-y_t)^2}{T-N}} \tag{4}$$

最近 N 期序列值的平均值作为未来各期的预测结果。一般 N 取值范围: $5 \leqslant N \leqslant 200$。当历史序列的基本趋势变化不大且序列中随机变动成分较多时，N 的取值应较大一些。否则 N 的取值应小一些。在有确定的季节变动周期的资料中，移动平均的项数应取周期长度。选择最佳 N 值的一个有效方法是，比较若干模型的预测误差。预测标准误差最小者为好。简单移动平均法只适合做近期预测，而且是预测目标的发展趋势变化不大的情况。如果目标的发展趋势存在其它的变化，采用简单移动平均法就会产生较大的预测偏差和滞后。

举个例子，某企业 1 月 ~11 月份的销售收入时间序列如表 2 示。试用一次简单滑动平均法预测第 12 月份的销售收入。

表 3: 销售收入时间序列表

月份 t	1	2	3	4	5	6
销售收入 y_t	533.8	574.6	606.9	649.8	705.1	772
月份 t	7	8	9	10	11	
销售收入 y_t	816.4	892.7	963.9	1015.1	1102.7	

解：分别取 N =4,N =5 的预测公式

$$\hat{y}_{t+1}^{(1)} = \frac{y_t + y_{t-1} + y_{t-2} + y_{t-3}}{4}, t = 4, 5, \cdots, 11$$

$$\hat{y}_{t+1}^{(2)} = \frac{y_t + y_{t-1} + y_{t-2} + y_{t-3}}{5}, t = 5, \cdots, 11$$

当 N=4 时，预测值 $\hat{y_{12}^{(1)}} = 993.6$，预测的标准误差为

$$S_1 = \sqrt{\frac{\sum\limits_{t=5}^{11}(\hat{y}_t^{(1)} - y_t)^2}{11-4}} = 150.5$$

当 N=5 时，预测值 $\hat{y_{12}^{(2)}} = 182.4$，预测的标准误差为

$$S_2 = \sqrt{\frac{\sum_{t=6}^{11}(\hat{y}_t^{(2)} - y_t)^2}{11-5}} = 958.2$$

计算结果表明，N=4 时，预测的标准误差较小，所以选取 N=4. 预测的第 12 月销售收入为 993.6。

5.2 加权移动平均法

在简单移动平均公式中，每期数据在求平均时的作用是等同的。但是，每期数据所包含的信息量不一样，近期数据包含着更多关于未来情况的信息。因此，把各期数据等同看待是不尽合理的，应考虑各期数据的重要性，对近期数据给予较大的权重，这就是加权移动平均法的基本思想。

设时间序列为 $y_1, y_2, \cdots, y_t, \cdots$; 加权移动平均公式为

$$M_{tw} = \frac{w_1 y_t + w_2 y_2 + \cdots + w_N y_{t-N+1}}{w_1 + w_2 + \cdots + w_N} \tag{5}$$

利用加权移动平均数来做预测，其预测公式为

$$\hat{y}_{t+1} = M_{tw} \tag{6}$$

即以第 t 期加权移动平均数作为第 t +1 期的预测值。在加权移动平均法中，w_t 的选择，同样具有一定的经验性。一般的原则是: 近期数据的权数大，远期数据的权数小。至于大到什么程度和小到什么程度，则需要按照预测者对序列的了解和分析来确定。

5.3 趋势移动平均法

简单移动平均法和加权移动平均法，在时间序列没有明显的趋势变动时，能够准确反映实际情况。但当时间序列出现直线增加或减少的变动趋势时，用简单移动平均法和加权移动平均法来预测就会出现滞后偏差。因此，需要进行修正，修正的方法是作二次移动平均，利用移动平均滞后偏差的规律来建立直线趋势的预测模型。这就是趋势移动平均法。一次移动的平均数为:

$$M_t^{(1)} = \frac{1}{N}(y_t + y_{t-1} + \cdots + y_{t-N+1})$$

在一次移动平均的基础上再进行一次移动平均就是二次移动平均，其计算公式为

$$M_t^{(2)} = \frac{1}{N}(M_t^{(1)} + \cdots + M_{t-N+1}^{(1)}) = M_{t-1}^{(2)} + \frac{1}{N}(M_t^{(1)} - M_{t-N}^{(1)})$$

下面讨论如何利用移动平均的滞后偏差建立直线趋势预测模型。设时间序列 $\{y_t\}$ 从某时期开始具有直线趋势，且认为未来时期也按此直线趋势变化，则可设此直线趋势预测模型为:

$$\hat{y}_{t+m} = a_t + b_t m, \quad m = 1, 2, \cdots$$

其中 t 为当前时期数; m 为由 t 至预测期的时期数; a_t 为截距; b_t 为斜率。两者又称为平滑系数。举个例子，假如我们有六个观测到的数据分别为

$$2, \quad 4, \quad 6, \quad 8, \quad 10, \quad 12.$$

可由平滑系数公式：

$$\begin{cases} a_t = 2M_t^{(1)} - M_t^{(2)} \\ b_t = \frac{2}{N-1}(M_t^{(1)} - M_t^{(2)}) \end{cases}$$

分别计算这条直线的斜率、截距。值得注意的是，这种做法会大幅度减少可用的时间序列个数，在进行拟合、预测时需要大量的数据才可以进行。例如：我国 1965~1985 年的发电总量如下表所示，试预测 1986 年和 1987 年的发电总量。

表 4: 我国发电量及一、二次移动平均值计算表

年份	t	发电总量 y_t	一次移动平均，N=6	二次移动平均，N=6
1965	1	676		
1966	2	825		
1967	3	774		
1968	4	716		
1969	5	940		
1970	6	1159	848.3	
1971	7	1384	966.3	
1972	8	1524	1082.8	
1973	9	1668	1231.8	
1974	10	1688	1393.8	
1975	11	1958	1563.5	1181.1
1976	12	2031	1708.8	1324.5
1977	13	2234	1850.5	1471.9
1978	14	2566	2024.2	1628.8
1979	15	2820	2216.2	1792.8
1980	16	3006	2435.8	1966.5
1981	17	3093	2625	2143.4
1982	18	3277	2832.7	2330.7
1983	19	3514	3046	2530
1984	20	3770	3246.7	2733.7
1985	21	4107	3461.2	2941.2

通过对数据可视化，可见发电总量基本呈直线上升趋势，可用趋势移动平均法来预测。取 N =

6 ，分别计算一次和二次移动平均值并列于表中。

$$M_{21}^{(1)} = 3461.2, \quad M_{21}^{(2)} = 2941.2$$

代入平滑系数公式，得:

$$a_{21} = 2M_{21}^{(1)} - M_{21}^{(2)} = 3981.1$$
$$b_{21} = \frac{2}{6-1}(M_{21}^{(1)} - M_{21}^{(2)}) = 208$$

于是可以得到 $t = 21$ 时直线趋势预测模型为

$$\hat{y}_{21+m} = 3981.1 + 208m$$

预测未来两年分别为 4192.1 和 4397.1.

趋势移动平均法对于同时存在直线趋势与周期波动的序列，是一种既能反映趋势变化，又可以有效地分离出来周期变动的方法。

5.4 指数平滑法

一次移动平均实际上认为最近 N 期数据对未来值影响相同，都加权 $\frac{1}{N}$ ；而 N 期以前的数据对未来值没有影响，加权为 0。但是，二次及更高次移动平均数的权数却不是 $\frac{1}{N}$ ，且次数越高，权数的结构越复杂，但永远保持对称的权数，即两端项权数小，中间项权数大，不符合一般系统的动态性。一般说来历史数据对未来值的影响是随时间间隔的增长而递减的。所以，更切合实际的方法应是对各期观测值依时间顺序进行加权平均作为预测值。指数平滑法可满足这一要求，而且具有简单的递推形式。指数平滑法根据平滑次数的不同，又分为一次指数平滑法、二次指数平滑法和三次指数平滑法等。

5.4.1 一次指数平滑法

设时间序列为 $y_1, y_2, \cdots, y_t, \cdots, \alpha$ 为加权系数，0<α<1，一次指数平滑公式为:

$$S_t^{(1)} = \alpha y_t + (1-\alpha)S_{t-1}^{(1)} = S_{t-1}^{(1)} + \alpha(y_t - S_{t-1}^{(1)})$$

这是由移动平均公式改进而来的，移动平均数的递推公式为

$$M_t^{(1)} = M_{t-1}^{(1)} + \frac{y_t - y_{t-N}}{N}$$

以 $M_{t-1}^{(1)}$ 作为 y_{t-N} 的最佳估计，将一次指数平滑公式展开，有:

$$S_t^{(1)} = \alpha y_t + (1-\alpha)[\alpha y_{t-1} + (1-\alpha)S_{t-2}^{(1)}] = \alpha\sum_{j=0}^{\infty}(1-\alpha)^j y_{t-j}$$

该式表明，$S_t^{(1)}$ 是全部历史数据的加权平均，加权系数分别为 $\alpha, \alpha(1-\alpha), \alpha(1-\alpha)^2, \cdots$; 显然有

$$\sum_{j=0}^{\infty}(1-\alpha)^j = \frac{\alpha}{1-(1-\alpha)} = 1$$

由于加权系数符合指数规律，又具有平滑数据的功能，故称为指数平滑。以这种平滑值进行预测，就是一次指数平滑法。预测模型为

$$\hat{y}_{t+1} = S_t^{(1)}$$

即:

$$\hat{y}_{t+1} = \alpha y_t + (1-\alpha)\hat{y}_t$$

也就是以第 t 期指数平滑值作为 t +1 期预测值。

在进行指数平滑时，加权系数的选择是很重要的。由上式可以看出，α 的大小规定了在新预测值中新数据和原预测值所占的比重。α 值越大，新数据所占的比重就愈大，原预测值所占的比重就愈小，反之亦然。若把公式改写为

$$\hat{y}_{t+1} = \hat{y}_t + \alpha(y_t - \hat{y}_t)$$

则从上式可看出，新预测值是根据预测误差对原预测值进行修正而得到的。α 的大小则体现了修正的幅度，α 值愈大，修正幅度愈大;α 值愈小，修正幅度也愈小。若选取 α =0，则 $y_{t+1} = y_t$，即下期预测值就等于本期预测值，在预测过程中不考虑任何新信息; 若选取 α=1，则 $y_{t+1} = y_t$ ，即下期预测值就等于本期观测值，完全不相信过去的信息。这两种极端情况很难做出正确的预测。因此，α 值应根据时间序列的具体性质在 0~1 之间选择。具体如何选择一般可遵循下列原则:

1. 如果时间序列波动不大，比较平稳，则 α 应取小一点，如 (0.1~0.5)。以减少修正幅度，使预测模型能包含较长时间序列的信息;
2. 如果时间序列具有迅速且明显的变动倾向，则 α 应取大一点，如 (0.6~0.8)。使预测模型灵敏度高一些，以便迅速跟上数据的变化。

在实用上，类似移动平均法，多取几个 α 值进行试算，看哪个预测误差小，就采用哪个。

用一次指数平滑法进行预测，除了选择合适的 α 外，还要确定初始值 $s_0^{(1)}$ 。初始值是由预测者估计或指定的。当时间序列的数据较多，比如在 20 个以上时，初始值对以后的预测值影响很少，可选用第一期数据为初始值。如果时间序列的数据较少，在 20 个以下时，初始值对以后的预测值影响很大，这时，就必须认真研究如何正确确定初始值。一般以最初几期实际值的平均值作为初始值。例如，某市 1976~1987 年某种电器销售额如表 4 所示。试预测 1988 年该电器销售额。

表 5: 某种电器销售额及指数平滑预测值计算表（单位：万元）

年份	t	实际销售额 y_t	预测值 $\hat{y}_t, \alpha = 0.2$	预测值 $\hat{y}_t, \alpha$ =0.5	预测值 $\hat{y}_t, \alpha$ =0.8
1976	1	50	51	51	51
1977	2	52	50.8	50.5	50.2
1978	3	47	51.04	51.25	51.64
1979	4	51	50.23	49.13	47.93
1980	5	49	50.39	50.06	50.39
1981	6	48	50.11	49.53	49.28
1982	7	51	49.69	48.77	48.26
1983	8	40	49.95	49.88	50.45
1984	9	48	47.96	44.94	42.09
1985	10	52	47.97	46.47	46.82
1986	11	51	48.77	49.24	50.96
1987	12	59	49.22	50.12	50.99

采用指数平滑法，并分别取 $\alpha = 0.2$,0.5 和 0.8 进行计算，初始值

$$S_0^{(1)} = \frac{y_1 + y_2}{2} = 51, \quad \text{即：} \hat{y}_1 = S_0^{(1)} = 51$$

按照预测模型 $\hat{y}_{t+1} = \alpha y_t + (1-\alpha)\hat{y}_t$，从表中可以看出，$\alpha$= 0.2,0.5 和 0.8 时，预测值是很不相同的。究竟 α 取何值为好，可通过计算它们的预测标准误差 S ，选取使 S 较小的那个 α 值。预测的标准误差为：

表 6: 预测的标准误差

α	0.2	0.5	0.8
S	4.5029	4.5908	4.8426

计算结果表明:α =0.2 时，S 较小，故选取 α =0.2，预测 1988 年该电器销售额为 y_{1988} =51.1754。

5.4.2 二次指数平滑法

一次指数平滑法虽然克服了移动平均法的缺点。但当时间序列的变动出现直线趋势时，用一次指数平滑法进行预测，仍存在明显的滞后偏差。因此，也必须加以修正。修正的方法与趋势移动平均法相同，即再作二次指数平滑，利用滞后偏差的规律建立直线趋势模型。这就是二次指数平滑法。其计算公式为

$$S_t^{(1)} = \alpha y_t + (1-\alpha) S_{t-1}^{(1)}$$

$$S_t^{(2)} = \alpha S_t^{(1)} + (1-\alpha) S_{t-1}^{(2)}$$

式中 $S_t^{(1)}$ 为一次指数的平滑值；$S_t^{(2)}$ 为二次指数的平滑值。当时间序列 $\{y_t\}$，从某时期开始具有直线趋势时，类似趋势移动平均法，可用直线趋势模型：$\hat{y}_{t+m} = a_t + b_{tm}, m = 1, 2, \cdots$，计算公式为

$$\begin{cases} a_t = 2S_t^{(1)} - S_t^{(2)} \\ b_t = \frac{\alpha}{1-\alpha}(S_t^{(1)} - S_t^{(2)}) \end{cases}$$

仍以我国 1965~1985 年的发电总量资料为例，试用二次指数平滑法预测 1986 年和 1987 年的发电总量。

取 $\alpha = 0.3$，初始值 $S_0^{(1)}$ 和 $S_0^{(2)}$ 都取序列的首项数值，即 $S_0^{(1)} = S_0^{(2)}$=676，计算 $S_0^{(1)}, S_0^{(2)}$ 列于表中。得到 $S_{21}^{(1)}$=3523.1，$S_{21}^{(2)}$ =3032.6，代入系数公式：

$$\begin{cases} a_t = 2S_t^{(1)} - S_t^{(2)} = 4013.7 \\ b_t = \frac{\alpha}{1-\alpha}(S_t^{(1)} - S_t^{(2)}) = 210.24 \end{cases}$$

于是，得 t = 21 时直线趋势方程为：y_{21+m} =4013.7+210.24m，预测 1986 年和 1987 年的发电总量为 (单位: 亿度)

$$y_{1986} = y_{22} = y_{21+1} = 4223.95$$

$$y_{1987} = y_{23} = y_{21+2} = 4434.19$$

表 7: 我国发电量及一、二次移动平均值计算表

年份	t	发电总量 y_t	一次平滑值	二次平滑值	y_{t+1} 的估计值
1965	1	676	676	676	
1966	2	825	720.7	689.4	676
1967	3	774	736.7	703.6	765.4
1968	4	716	730.5	711.7	784.0
1969	5	940	793.3	736.2	757.4
1970	6	1159	903.0	786.2	875.0
1971	7	1384	1047.3	864.6	1069.9
1972	8	1524	1190.3	962.3	1308.4
1973	9	1668	1333.6	1073.7	1516.1
1974	10	1688	1439.9	1183.6	1705.0
1975	11	1958	1595.4	1307.1	1806.1
1976	12	2031	1726.1	1432.8	2007.2
1977	13	2234	1878.4	1566.5	2145.0
1978	14	2566	2084.7	1722.0	2324.1
1979	15	2820	2305.3	1897.0	2602.9
1980	16	3006	2515.5	2082.5	2888.6
1981	17	3093	2688.8	2264.4	3134.1
1982	18	3277	2865.2	2444.6	3295.0
1983	19	3514	3059.9	2629.2	3466.1
1984	20	3770	3272.9	2822.3	3675.1
1985	21	4107	3523.1	3032.6	3916.6

5.4.3 三次指数平滑法

当时间序列的变动表现为二次曲线趋势时，则需要用三次指数平滑法。三次指数平滑是在二次指数平滑的基础上，再进行一次平滑，其计算公式为

$$\begin{cases} S_t^{(1)} = \alpha y_t + (1-\alpha) S_{t-1}^{(1)} \\ S_t^{(2)} = \alpha S_t^{(1)} + (1-\alpha) S_{t-1}^{(2)} \\ S_t^{(3)} = \alpha S_t^{(2)} + (1-\alpha) S_{t-1}^{(3)} \end{cases}$$

式中 $S_{t-1}^{(3)}$ 为三次指数平滑值。三次指数平滑法的预测模型为: $\hat{y}_{t+m}=a_t+b_tm+C_tm^2, m=1,2,\cdots$, 其中

$$\begin{cases} a_t = 3S_t^{(1)}-3S_t^{(2)}+S_t^{(3)} \\ b_t = \frac{\alpha}{2(1-\alpha)^2}[(6-5\alpha)S_t^{(1)}-2(5-4\alpha)S_t^{(2)}+(4-3\alpha)S_t^{(3)}] \\ c_t = \frac{\alpha^2}{2(1-\alpha)^2}[S_t^{(1)}-2S_t^{(2)}+S_t^{(3)}] \end{cases}$$

对数据进行可视化，可见投资总额呈二次曲线上升，可用三次指数平滑法进行预测。取 α=0.3，初始值 $S_1^{(0)}=S_2^{(0)}=S_3^{(0)}=\frac{y_1+y_2+y_3}{3}=21.94$，计算 $S_t^{(1)}, S_t^{(2)}, S_t^{(3)}$ 列于表中，得到得到 $S_{11}^{(1)}=151.77$ $S_{11}^{(2)}=101.28$ $S_{11}^{(3)}=68.43$，可得到当 t =11 时，$a_{11}=219.91, b_{11}=38.38, c_{11}=1.62$，于是，得 t =11 时预测模型为 $\hat{y}_{11+m}=219.91+38.38m+1.62m^2$，预测 1989 年和 1990 年的固定资产投资总额为 (单位: 亿元):

$$\hat{y}_{1989}=\hat{y}_{12}=\hat{y}_{11+1}=a_{11}+b_{11}+c_{11}=259.91$$

$$\hat{y}_{1990}=\hat{y}_{13}=\hat{y}_{11+2}=a_{11}+2b_{11}+2^2c_{11}=303.16$$

因为国家从 1989 年开始对固定资产投资采取压缩政策，这些预测值显然偏高，应作适当的修正，以消除政策因素的影响。

指数平滑预测模型是以时刻 t 为起点，综合历史序列的信息，对未来进行预测的。选择合适的加权系数 α 是提高预测精度的关键环节。根据实践经验，α 的取值范围一般以 0.1~0.3 为宜。α 值愈大，加权系数序列衰减速度愈快，所以实际上 α 取值大小起着控制参加平均的历史数据的个数的作用。α 值愈大意味着采用的数据愈少。因此，可以得到选择 α 值的一些基本准则。

1. 如果序列的基本趋势比较稳，预测偏差由随机因素造成，则 α 值应取小一些，以减少修正幅度，使预测模型能包含更多历史数据的信息。

2. 如果预测目标的基本趋势已发生系统的变化，则 α 值应取得大一些。这样，可以偏重新数据的信息对原模型进行大幅度修正，以使预测模型适应预测目标的新变化。

另外，由于指数平滑公式是递推计算公式，所以必须确定初始值 $S_0^{(1)}, S_0^{(2)}, S_0^{(3)}$。可以取前 3~5 个数据的算术平均值作为初始值。

5.5 差分指数平滑法

在上节我们已经讲过，当时间序列的变动具有直线趋势时，用一次指数平滑法会出现滞后偏差，其原因在于数据不满足模型要求。因此，我们也可以从数据变换的角度来考虑改进措施，即在运用指数平滑法以前先对数据作一些技术上的处理，使之能适合于一次指数平滑模型，以后再对输出结果作技术上的返回处理，使之恢复为原变量的形态。差分方法是改变数据变动趋势的简易方法。下面我们讨论如何用差分方法来改进指数平滑法。

在前面我们已分析过，指数平滑值实际上是一种加权平均数。因此把序列中逐期增量的加权平均数 (指数平滑值) 加上当前值的实际数进行预测，比一次指数平滑法只用变量以往取值的加权平均数作为下一期的预测更合理。从而使预测值始终围绕实际值上下波动，从根本上解决了在有直线增长趋势的情况下，用一次指数平滑法所得出的结果始终落后于实际值的问题。

差分方法和指数平滑法的联合运用，除了能克服一次指数平滑法的滞后偏差之外，对初始值的

问题也有显著的改进。因为数据经过差分处理后，所产生的新序列基本上是平稳的。这时，初始值取新序列的第一期数据对于未来预测值不会有多大影响。其次，它拓展了指数平滑法的适用范围，使一些原来需要运用配合直线趋势模型处理的情况可用这种组合模型来取代。但是，对于指数平滑法存在的加权系数 的选择问题，以及只能逐期预测问题，差分指数平滑模型也没有改进。

5.6 自适应滤波法

自适应滤波法与移动平均法、指数平滑法一样，也是以时间序列的历史观测值进行某种加权平均来预测的，它要寻找一组“最佳”的权数，其办法是先用一组给定的权数来计算一个预测值，然后计算预测误差，再根据预测误差调整权数以减少误差。这样反复进行，直至找出一组“最佳”权数，使误差减少到最低限度。由于这种调整权数的过程与通讯工程中的传输噪声过滤过程极为接近，故称为自适应滤波法。自适应滤波法的基本预测公式为

$$\hat{y}_{t+1} = w_1 y_t + w_2 y_{t-1} + \cdots + w_N y_{t-N+1} = \sum_{i=1}^{N} w_i y_{t-i+1}$$

$\hat{y}_{t+1}$ 为第 $t+1$ 期的预测值，w_i 为第 $t-i+1$ 期的观测值权数，y_{t-i+1} 为第 $t-i+1$ 期的观测值，N 为权数的个数。其调整权数的公式为

$$w_i' = w_i + 2k \cdot e_{i+1} y_{t-i+1}$$

式中，$i = 1, 2, \cdots\ t = N, N+1, \cdots, n, n$ 为序列数据的个数，w_i 为调整前的第 i 个权数，w' 为调整后的第 i 个权数，k 为学习常数，e_{i+1} 为第 $t+1$ 期的预测误差。上式表明: 调整后的一组权数应等于旧的一组权数加上误差调整项，这个调整项包括预测误差、原观测值和学习常数等三个因素。学习常数 k 的大小决定权数调整的速度。例如，下面有 10 期数据，试用自适应滤波法，以两个权数来求第 11 期的预测值。

表 8: 某时间序列数据表

时期 t	1	2	3	4	5	6	7	8	9	10
观测值 y_t	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

本例中 $N = 2$。取初始权数 $w_1 = 0.5$，$w_2 = 0.5$，并设 $k = 0.9$。t 的取值由 $N = 2$ 开始，当 t = 2 时:

- 按预测公式，求第 $t+1 = 3$ 期的预测值。$\hat{y}_{t+1} = \hat{y}_3 = w_1 y_2 + w_2 y_1 = 0.15$
- 计算预测误差。$e_{t+1} = e_3 = y_3 - \hat{y}_3 = 0.3 - 0.15 = 0.15$
- 根据调整权数公式：$w_1' = w_1 + 2k e_3 y_2 = 0.554$，$w_2' = w_2 + 2k e_3 y_1 = 0.527$

上面我们完成了第一次轮调整，把现有的新权数作为初始权数，重新开始 $t = 2$ 的过程。这样反复进行下去，到预测误差 (指新一轮的预测总误差) 没有明显改进时，就认为获得了一组“最佳”权数，能实际用来预测第 11 期的数值。本例在调整过程中，可使得误差降为零，而权数达到稳定不变，最后得到的“最佳”权数为

$$w_1' = 2.0, \quad w_2' = -1.0$$

用“最佳”权数预测第 11 期的取值

$$\hat{y}_{11} = w_1' y_{10} + w_2' y_9 = 1.1$$

在实际应用中，权数调整计算工作量可能很大，必须借助于计算机才能实现。

在开始调整权数时，首先要确定权数个数 N 和学习常数 k 。一般说来，当时间序列的观测值呈季节变动时，N 应取季节性长度值。如序列以一年为周期进行季节变动时，若数据是月度的，则取 $N = 12$ ，若季节是季度的，则取 $N = 4$ 。如果时间序列无明显的周期变动，则可用自相关系数法来确定，即取 N 为最高自相关系数的滞后时期。

k 的取值一般可定为 $1/N$ ，也可以用不同的 k 值来进行计算，以确定一个能使 S 最小的 k 值。

初始权数的确定也很重要，如无其它依据，也可用 $1/N$ 作为初始权系数用，即

$$w_i = \frac{1}{N}(i = 1, 2, \cdots, N)$$

自适应滤波法有两个明显的优点: 一是技术比较简单，可根据预测意图来选择权数的个数和学习常数，以控制预测。也可以由计算机自动选定。二是它使用了全部历史数据来寻求最佳权系数，并随数据轨迹的变化而不断更新权数，从而不断改进预测。

由于自适应滤波法的预测模型简单，又可以在计算机上对数据进行处理，所以这种预测方法应用较为广泛。

5.7 平稳时间序列

这里的平稳是指宽平稳，其特性是序列的统计特性不随时间的平移而变化，即均值和协方差不随时间的平移而变化[1]。

5.7.1 平稳随机序列 (平稳时间序列)

定义 1: 给定随机过程 $\{X_t, t \in T\}$。固定 t, X_t 是一个随机变量，设其均值为 μ_t ，当 t 变动时，此均值是 t 的函数，记为

$$\mu_t = E(X_t)$$

称为随机过程的均值函数。

固定 t，设 X_t 的方差为 σ_t^2。当 t 变动时，这个方差也是 t 的函数，记为

$$\sigma_t^2 = Var(X_t) = E[(X_t - \mu_t)^2]$$

称为随机过程的**方差函数**。方差函数的平方根 σ_t 称为随机过程的**标准差函数**，它表示随机过程 X_t 对于均值函数 μ_t 的偏离程度。

定义 2: 对随机过程 $\{X_t, t \in T\}$，取定 $t, s \in T$，定义其自协方差函数为

$$\gamma_{t,s} = Cov(X_t, X_s) = E[(X_t - \mu_t)(X_s - \mu_s)]$$

为刻画 $\{X_t, t \in T\}$ 在时刻 t 与 s 之间的相关性，还可将 $\gamma_{t,s}$ 标准化，即定义自相关函数

$$\rho_{t,s} = \frac{\gamma_{t,s}}{\sqrt{\gamma_{t,t}}\sqrt{\gamma_{s,s}}} = \frac{\gamma_{t,s}}{\rho_t \rho_s}$$

[1] 本笔记只着重探讨时间序列的基本含义，更多知识请参考时间序列相关教材

因此，自相关函数 $\rho_{t,s}$ 是标准化自协方差函数。

定义 3：设随机序列 $\{X_t, t=0,\pm1,\pm2,\cdots\}$ 满足

- $E(X_t)=\mu=$ 常数
- $\gamma_{t+k,t}=\gamma_k(k=0,\pm1,\pm2,\cdots)$ 与 t 无关

则称 X_t 为平稳随机序列 (平稳时间序列)，简称**平稳序列**。

定义 4：设平稳序列 $\{\varepsilon_t, t=0,\pm1,\pm2,\cdots\}$ 的自协方差函数 γ_k 是

$$\gamma_k=\sigma^2\delta_{k,0}=\begin{cases}0, & k\neq 0\\ \sigma^2, & k=0\end{cases}$$

其中 $\delta_{k,0}$ 当 $k=0$ 时为 1，当 $k\neq 1$ 时为 0，则称该序列为**平稳白噪声序列**。

平稳白噪声序列的方差是常数 σ^2 ，因为 $\gamma_k=0(k\neq 0)$，则 ε_t 的任意两个不同时点之间是不相关的。平稳白噪声序列是一种最基本的平稳序列。

既然平稳白噪声序列各种统计参数都不变，那么我们如何分辨一列数据是否是白噪声还是真正的时间序列? 这里我们用到时间序列的另一个特性：前后相关性，白噪声是没有前后相关性的，因此我们借此来进行检验：即 Daniel 检验。Daniel 检验方法建立在 Spearman 相关系数的基础上。

Spearman 相关系数是一种秩相关系数。先讲样本的秩的概念。设 $x_1,x_2,\cdots,x_n$ 是从一元总体抽取的容量为 n 的样本，其顺序统计量是 $x_{(1)},x_{(2)},\cdots,x_{(n)}$。若 $x_i=x_{(k)}$，则称 k 是 x_i 在样本中的**秩**，记作 R_i ，对每一个 $i=1,2,\cdots,n$ ，称 R_i 是第 i 个秩统计量。$R_1,R_2,\cdots,R_n$ 总称为**秩统计量**。例如，对样本数据

$$-0.8,\quad -3.1,\quad 1.1,\quad -5.2,\quad 4.2$$

顺序统计量是

$$-5.2,\quad -3.1,\quad -0.8,\quad 1.1,\quad 4.2$$

而秩统计量是

$$3,\quad 2,\quad 4,\quad 1,\quad 5$$

你可认为：秩就是“秩序”，就是原来样本数据在顺序统计量中的排位数，例如-0.8 在顺序排位中排名第 3，因此原数据的秩统计量为 3，以此类推。对于二维总体 (X,Y) 的样本观测数据 $(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)$，可得各分量 X,Y 的一元样本数据 $x_1,x_2,\cdots,x_n$ 与 $y_1,y_2,\cdots,y_n$ 。设 $x_1,x_2,\cdots,x_n$ 的秩统计量是

$$R_1,\quad R_2,\quad \cdots,\quad R_n$$

$y_1,y_2,\cdots,y_n$ 的秩统计量是

$$S_1,\quad S_2,\quad \cdots,\quad S_n$$

当 X,Y 联系比较紧密时，这两组秩统计量联系也是紧密的。Spearman 相关系数定义为这两组秩统计量的相关系数，即 Spearman 相关系数是

$$q_{xy}=\frac{\sum\limits_{i=1}^{n}\left(R_i-\bar{R}\right)\left(S_i-\bar{S}\right)}{\sqrt{\sum\limits_{i=1}^{n}\left(R_i-\bar{R}\right)^2}\sqrt{\sum\limits_{i=1}^{n}\left(S_i-\bar{S}\right)^2}}$$

其中 $\bar{R}=\frac{1}{n}\sum\limits_{i=1}^{n}R_i,\bar{S}=\frac{1}{n}\sum\limits_{i=1}^{n}S_i$，经过运算，可证

$$q_{xy}=1-\frac{6}{n\left(n^2-1\right)}\sum_{i=1}^{n}d_i^2$$

其中 $d_i=R_i-S_i,\quad i=1,2,\cdots,n$。

对于 Spearman 相关系数，作假设检验

$$H_0:\rho_{XY}=0,\quad H_1:\rho_{XY}\neq 0$$

可以证明，当 (X,Y) 是二元正态总体，且 H_0 成立时，统计量

$$t=\frac{q_{xy}\sqrt{n-2}}{\sqrt{1-q_{xy}^2}}$$

服从自由度为 $n-2$ 的 t 分布 $t(n-2)$。

对于给定的显著水平 α，通过 t 分布表可查到统计量 t 的临界值 $t_{\alpha/2}(n-2)$ ，当 $t\leqslant t_{\alpha/2}(n-2)$ 时，接受 H_0; 当 $t>t_{\alpha/2}(n-2)$ 时，拒绝 H_0。

对于时间序列的样本 $X_1,X_2,\cdots,X_n$，记 X_t 的秩为 $R_t=R(X_t)$，考虑变量对 $(t,R_t),t=1,2,\cdots,n$ 的 Spearman 相关系数 q_s，有

$$q_s=1-\frac{6}{n\left(n^2-1\right)}\sum_{i=1}^{n}\left(t-R_t\right)^2$$

构造统计量

$$T=\frac{q_s\sqrt{n-2}}{\sqrt{1-q_s^2}}$$

作下列假设检验:

$$H_0:\text{序列}X_t\text{平稳},\quad H_1:\text{序列}X_t\text{非平稳 (存在上升或下降趋势)}$$

Daniel 检验方法: 对于显著水平 α，由时间序列 X_t 计算 $(t,R_t),t=1,2,\cdots,n$ 的 Spearman 秩相关系数 q_s，若 $|T|>t_{\alpha/2}(n-2)$，则拒绝 H_0，认为序列非平稳。且当 $q_s>0$ 时，认为序列有上升趋势;$q_s<0$ 时，认为序列有下降趋势。又当 $|T|\leqslant t_{\alpha/2}(n-2)$ 时，接受 H_0 ，可以认为 X_t 是平稳序列。

5.7.2 平稳序列自协方差函数与自相关函数的估计

对平稳序列 X_t 的样本 $X_1,X_2,\cdots,X_n$，可以用样本均值估计随机序列的均值

$$\hat{\mu}=\frac{1}{n}\sum_{i=1}^{n}X_t=\bar{X}$$

γ_k 通常有如下两种估计

$$\hat{\gamma}_k=\frac{1}{n}\sum_{t=1}^{n-k}\left(X_{t+k}-\bar{X}\right)\left(X_t-\bar{X}\right),0\leq k\leq K$$

且 $\hat{\gamma}_{-k}=\hat{\gamma}_k$，或者

$$\hat{\gamma}_k=\frac{1}{n-k}\sum_{t=1}^{n-k}\left(X_{t+k}-\bar{X}\right)\left(X_t-\bar{X}\right),0\leq k\leq K$$

ρ_k 的估计为

$$\hat{\rho}_k=\frac{\hat{\gamma}_k}{\hat{\gamma}_0},0\leq k\leq K$$

5.8 ARMA 时间序列

下面介绍一种重要的平稳时间序列—ARMA 时间序列。ARMA 时间序列分为三种类型:

- AR 模型，即自回归序列 (Auto Regressive Model);
- MA 序列，即滑动平均序列 (Moving Average Model);
- ARMA 序列，即自回归滑动平均序列 (Auto Regressive Moving Average Model)。

可证 ARMA 时间序列具有遍历性，因此可以通过它的一个样本估计自协方差函数及自相关函数。

5.8.1 AR(p) 序列

设 $\{X_t, t = 0, \pm 1, \pm 2, \cdots\}$ 是零均值平稳序列，满足下列模型:

$$X_t = \varphi_1 X_{t-1} + \varphi_2 X_{t-2} + \cdots + \varphi_n X_{t-p} + \varepsilon_t$$

其中 ε_t 是零均值、方差是 σ_ε^2 的平稳白噪声。则称 X_t 是阶数为 p 的自回归序列，简记为 AR(p) 序列，而

$$\varphi = (\varphi_1, \varphi_2, \cdots, \varphi_p)^T$$

称为自回归参数向量，其分量 $\varphi_j, j = 1, 2, \cdots, p$ 称为**自回归系数**。

引进后移算子对描述式比较方便。算子 B 定义如下:

$$BX_t \equiv X_{t-1}, \quad B^k X_t \equiv X_{t-k}$$

记算子多项式

$$\varphi(B) = 1 - \varphi_1 B - \varphi_2 B^2 - \cdots - \varphi_p B^p$$

则 AR(p) 可以改写为

$$\varphi(B) X_t = \varepsilon_t$$

5.8.2 MA(q) 序列

设 $\{X_t, t = 0, \pm 1, \pm 2, \cdots\}$ 是零均值平稳序列，满足下列模型:

$$X_t = \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q},$$

其中 ε_t 是零均值、方差是 σ_ε^2 的平稳白噪声。则称 X_t 是阶数为 q 的滑动平均序列，简记为 MA(q) 序列，而

$$\theta = (\theta_1, \theta_2, \cdots, \theta_q)^T$$

称为滑动平均参数向量，其分量 $\theta_j, j = 1, 2, \cdots, q$ 称为**滑动平均系数**。

对于线性后移算子 B ，有

$$B\varepsilon_t \equiv \varepsilon_{t-1}, \quad B^k \varepsilon_t \equiv \varepsilon_{t-k};$$

再引进算子多项式

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q$$

则 MA(q) 可以改写为

$$X_1 = \theta(B)\varepsilon_1$$

5.8.3 ARMA(p,q) 序列

设 $\{X_t, t=0,\pm1,\pm2,\cdots\}$ 是零均值平稳序列，满足下列模型:

$$X_t - \varphi_1 X_{t-1} - \cdots - \varphi_p X_{t-p} = \varepsilon_t - \theta_1 \varepsilon_{t-1} - \cdots - \theta_q \varepsilon_{t-q}$$

其中 ε_t 是零均值、方差是 σ_ε^2 的平稳白噪声，则称 X_t 是阶数为 p,q 的**自回归滑动平均序列**，简记为 ARMA(p,q) 。当 $q=0$ 时，它是 AR(p) 序列; 当 $p=0$ 时，它为 MA(q) 序列。

应用算子多项式 $\varphi(B), \theta(B)$，可以将上式写为

$$\varphi(B) X_t = \theta(B) \varepsilon_t$$

对于一般的平稳序列 $\{X_t, t=0,\pm1,\pm2,\cdots\}$，设其均值 $E(X_t)=\mu$，满足下列模型:

$$(X_t - \mu) - \varphi_1 (X_{t-1} - \mu) - \cdots - \varphi_p (X_{t-p} - \mu) = \varepsilon_t - \theta_1 \varepsilon_{t-1} - \cdots - \theta_q \varepsilon_{t-9}$$

其中 ε_t 是零均值、方差是 σ_ε^2 的平稳白噪声，利用后移算子 $\varphi(B), \theta(B)$，可表为:

$$\varphi(B)(X_t - \mu) = \theta(B) \varepsilon_t$$

关于算子多项式 $\varphi(B), \theta(B)$ 通常还要作下列假定:

- $\varphi(B)$ 和 $\theta(B)$ 无公共因子，又 $\varphi_p \neq 0, \theta_q \neq 0$
- $\varphi(B)=0$ 的根全在单位圆外，这一条件称为模型的平稳性条件;
- $\theta(B)=0$ 的根全在单位圆外，这一条件称为模型的可逆性条件。

5.9 ARMA 序列的相关特性

我们要阐述 AR，MA，ARMA 序列的相关特性，这些特性对于序列的建模起着重要的作用。相关特性包括相关函数，以及下面将要阐述的偏相关函数。

5.9.1 MA(q) 序列的自相关函数

设 $\{X_t, t=0,\pm1,\pm2,\cdots\}$ 是 MA(q) 序列

$$X_t = \theta(B)\varepsilon_t = \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q}$$

其中 ε_t 是零均值、方差是 σ_ε^2 的平稳白噪声，则 X_t 是零均值平稳序列，其自协方差函数为

$$\gamma_k = \begin{cases} \sigma_\varepsilon^2 \left(1 + \theta_1^2 + \theta_2^2 + \cdots + \theta_q^2\right), & k=0 \\ \sigma_\varepsilon^2 \left(-\theta_k + \theta_1\theta_{k+1} + \cdots + \theta_{q-k}\theta_q\right), & 1 \le k \le q \\ 0, & k > q \end{cases}$$

自相关函数为

$$\rho_k = \frac{\gamma_k}{\gamma_0} = \begin{cases} 1, & k=0 \\ \frac{-\theta_k + \theta_1\theta_{k+1} + \cdots + \theta_{q-k}\theta_q}{1 + \theta_1^2 + \theta_2^2 + \cdots + \theta_q^2}, & 1 \le k \le q \\ 0, & k > q \end{cases}$$

注意到，当 $k > q$ 时

$$\gamma_k = 0, \quad \rho_k = 0$$

这表明对于 MA(q) 序列，当 $t-s > q$ 时，X_t 与 X_s 不相关，这一性质称为自协方差函数与自相关函数的 **q 步截尾性**。

5.9.2 AR(p) 序列的自相关函数

设 X_t 是 AR(p) 序列

$$X_t = \varphi_1 X_{t-1} + \varphi_2 X_{t-2} + \cdots + \varphi_p X_{t-p} + \varepsilon_t$$

$\mathrm{Var}(\varepsilon_t) = \sigma_\varepsilon^2$，用 $X_{t-k}(k > 0)$ 乘上式两端，求均值得

$$E\left(X_t X_{t-k}\right) = \varphi_1 E\left(X_{t-1} X_{t-k}\right) + \varphi_2 E\left(X_{t-2} X_{t-k}\right) + \cdots + \varphi_p E\left(X_{t-p} X_{t-k}\right) + E\left(\varepsilon_t X_{t-k}\right)$$

且 AR 模型有一个极为重要的公式

$$E(X_{t-k}\varepsilon_t) = 0, k \leqslant 1$$

故得

$$\gamma_k = \varphi_1 \gamma_{k-1} + \varphi_2 \gamma_{k-2} + \cdots + \varphi_p \gamma_{k-p}, \quad k > 0,$$

即

$$\varphi(B)\gamma_k = 0, \quad k > 0$$

这是自协方差函数 γ_k 满足的差分方程。自相关函数 ρ_k 也满足同样的差分方程

$$\varphi(B)\rho_k = 0, \quad k > 0$$

5.9.3 ARMA(p, q) 序列的自相关函数

设 $\{X_t, t = 0, \pm 1, \pm 2, \cdots\}$ 是 ARMA(p, q) 序列

$$X_t - \varphi_1 X_{t-1} - \cdots - \varphi_p X_{t-p} = \varepsilon_t - \theta_1 \varepsilon_{t-1} - \cdots - \theta_q \varepsilon_{t-q}$$

则其自协方差函数满足

$$\varphi(B)\gamma_k = \begin{cases} 0, & k > q \\ -\theta_q \sigma_\varepsilon^2, & k = q \end{cases}$$

取 $k = q+1, q+2, \cdots, q+p$，得到方程组

$$\begin{cases} \gamma_{q+1} = \varphi_1 \gamma_q + \varphi_2 \gamma_{q-1} + \cdots + \varphi_p \gamma_{q+1-p} \\ \gamma_{q+2} = \varphi_1 \gamma_{q+1} + \varphi_2 \gamma_q + \cdots + \varphi_p \gamma_{q+2-p} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \gamma_{q+p} = \varphi_1 \gamma_{q+p-1} + \varphi_2 \gamma_{q+p-2} + \cdots + \varphi_p \gamma_q \end{cases}$$

将其写成矩阵形式有

$$\begin{bmatrix} \gamma_{q+1} \\ \gamma_{q+2} \\ \vdots \\ \gamma_{q+p} \end{bmatrix} = \begin{bmatrix} \gamma_q & \gamma_{q-1} & \cdots & \gamma_{q+1-p} \\ \gamma_{q+1} & \gamma_q & \cdots & \gamma_{q+2-p} \\ \vdots & \vdots & & \vdots \\ \gamma_{q+p-1} & \gamma_{q+p-2} & \cdots & \gamma_q \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_p \end{bmatrix}$$

解此方程组，可得自回归系数向量 $(\varphi_1, \varphi_2, \cdots, \varphi_p)^T$.

其次，令 $Y_t = \varphi(B)X_t = X_t - \varphi_1 X_{t-1} - \cdots - \varphi_p X_{t-p}$，则 $\{Y_t, t = 0, \pm 1, \pm 2, \cdots\}$ 是参数为 $(\theta_1, \theta_2, \cdots, \theta_q)^T$ 的 MA(q) 序列，$\{Y_t, t = 0, \pm 1, \pm 2, \cdots\}$ 的自协方差函数

$$\gamma_k^{(Y)} = E\left(Y_{t+k}Y_t\right) = E\left[\left(-\sum_{i=0}^{p}\varphi_i X_{t+k-i}\right)\left(-\sum_{j=0}^{p}\varphi_j X_{t-j}\right)\right] = \sum_{i,j=0}^{p}\varphi_i\varphi_j\gamma_{k+j-i}, \quad (\varphi_0 = -1)$$

即

$$\sum_{i,j=0}^{p}\varphi_i\varphi_j\gamma_{k+j-i} = \begin{cases} \sigma_\varepsilon^2\left(1 + \theta_1^2 + \theta_2^2 + \cdots + \theta_q^2\right), & k = 0 \\ \sigma_\varepsilon^2\left(-\theta_k + \theta_1\theta_{k+1} + \cdots + \theta_{q-k}\theta_q\right), & 1 \le k \le q \end{cases}$$

解此方程组，得滑动平均系数向量 $(\theta_1, \theta_2, \cdots, \theta_q)^T$ 及 σ_ε^2.

可以证明，ARMA(p, q) 序列的自协方差函数 (自相关函数) 为拖尾的。

5.9.4 偏相关函数及 AR(p) 序列偏相关函数的截尾性

设 $\{X_t, t = 0, \pm 1, \pm 2, \cdots\}$ 是零均值平稳序列从时间序列，从预报的角度引出偏相关函数的定义。如果已知 $\{X_{t-1}, X_{t-2}, \cdots, X_{t-k}\}$ 的值，要求对 X_t 作出预报。此时，可以考虑由 $\{X_{t-1}, X_{t-2}, \cdots, X_{t-k}\}$ 对 X_t 的线性最小均方估计，即选择系数 $\varphi_{k,1}, \varphi_{k,2}, \cdots, \varphi_{k,k}$，使得

$$\delta = E\left[\left(X_t - \sum_{j=1}^{k}\varphi_{k,j}X_{t-j}\right)^2\right] = \min$$

将 δ 展开，得

$$\delta = \gamma_0 - 2\sum_{j=1}^{k}\varphi_{k,j}\gamma_j + \sum_{j=1}^{k}\sum_{i=1}^{k}\varphi_{k,j}\varphi_{k,i}\gamma_{j-i}$$

令 $\frac{\partial\delta}{\partial\varphi_{k,j}} = 0, \quad j = 1, 2, \cdots, k$，得

$$-\gamma_j + \sum_{i=1}^{k}\varphi_{k,i}\gamma_{j-i} = 0, \quad j = 1, 2, \cdots, k$$

两端同除 γ_0 并写成矩阵形式，可知 $\varphi_{k,j}$ 应满足下列线性方程组

$$\begin{bmatrix} 1 & \rho_1 & \cdots & \rho_{k-1} \\ \rho_1 & 1 & \cdots & \rho_{k-2} \\ \vdots & \vdots & & \vdots \\ \rho_{k-1} & \rho_{k-2} & \cdots & 1 \end{bmatrix}\begin{bmatrix} \varphi_{k,1} \\ \varphi_{k,2} \\ \vdots \\ \varphi_{k,k} \end{bmatrix} = \begin{bmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_k \end{bmatrix}$$

该式称为 Yule-Walker 方程。

我们称 $\varphi_{k,k}, k = 1, 2, \cdots$ 为 X_t 的偏相关函数。。该式系数矩阵是 Toeplitz 矩阵，这一线性方程组的解 $\varphi_{k,1}, \varphi_{k,2}, \cdots, \varphi_{k,k}$ 有下列递推公式

$$\begin{cases} \varphi_{1,1} = \rho_1 \\ \varphi_{k+1,k+1} = \left(\rho_{k+1} - \sum_{j=1}^{k}\rho_{k+1-j}\varphi_{k,j}\right)\left(1 - \sum_{j=1}^{k}\rho_j\varphi_{k,j}\right)^{-1} \\ \varphi_{k+1,j} = \varphi_{k,j} - \varphi_{k+1,k+1}\varphi_{k,k+1-j}, \quad j = 1, 2, \cdots, k \end{cases}$$

AR(p) 序列的偏相关函数 $\{\varphi_{k,k}, k = 1, 2, \cdots\}$ 是 p 步截尾的，且有

$$\begin{cases} \varphi_{k,k} = \varphi_k, & 1 \le k \le p \\ \varphi_{k,k} = 0, & k > p \end{cases}$$

对于 MA(q) 和 ARMA(p, q) 序列，其偏相关函数 $\{\varphi_{k,k}, k = 1, 2, \cdots\}$ 具有拖尾性。

5.9.5 ARMA 时间序列的建模与预报

在实际问题中，若考察的时间序列是 ARMA 序列

$$\varphi(B)X_t = \theta(B)\varepsilon_t,$$

首先要进行模型的识别与定阶，即要判断是 AR(p) ，MA(q) ，ARMA(p, q) 模型的类别，并估计阶数 p, q 。其实，这都归结到模型的定阶问题。当模型定阶后，就要对模型参数 $\varphi = (\varphi_1, \varphi_2, \cdots, \varphi_p)^T$ 及 $\theta = (\theta_1, \theta_2, \cdots, \theta_q)^T$ 进行估计。定阶与参数估计完成后，还要对模型进行检验，即要检验 ε_t 是否为平稳白噪声。若检验获得通过，则 ARMA 时间序列的建模完成。作为时间序列建模之后的一个重要应用，我们还要讨论 ARMA 时间序列的预报。

6 机器学习算法

6.1 模型加载与保存

Matlab 中可以导入自己的.m 文件，Python 中也同样可以用这些进行数据模型的保存。sklearn 中训练完毕后，我们可以保存相应的 pkl 二进制文件，进而在下一次启动时直接使用，相关命令如下：

```
joblib.dump(rf,test' test.pkl' ) #保存
estimator = joblib.load( 'test.pkl' )#加载
```

6.2 决策树

1948 年，信息论创始人**香农**发表了跨时代的论文——《通信的数学原理》，并在其中阐述了信息的计算单位：**比特 (bit)**。如何直观理解比特？我们举一个简单的例子：假设某一年有 32 支队伍参加世界杯比赛，他们的编号分别对应着 1-32，在不知道任何球队信息的情况下，我们最多猜 $log_232 = 5$ 次就能知道哪支队伍获得了冠军，此时的信息大小一共是 5bit；若有 64 支队伍，我们需要 6bit 的信息就可以知道谁是冠军。如果我们知道了一点球队历史，得知巴西、德国和阿根廷三只队伍的获胜概率较大，皆为 $\frac{1}{8}$，那么我们猜测的次数一定比 5bit 要小。也就是说，在进行二分选择时，两侧的权重发生了变化，它们不再是 50% 与 50%。此时判断“谁是冠军”需要的信息量为:

$$Ent(D) = -\sum_{k=1}^{|y|} p + klog_2p_k$$

其中，$Ent(D)$ 为**信息熵**，单位为比特 (bit)，p_k 为第 k 支队伍获胜的概率，当且仅当 32 支队伍的获胜概率相同，即 $p_1 = p_2 = \cdots = p_k = \frac{1}{32}$ 时，所需信息为 $log_232 = 5bit$，达到最大。信息论中规定：信息熵越大，不确定性就越大，称 D 的**纯度**越小，判断越困难。我们用“信息增益”来量化纯度提升的大小:

$$Gain(D, a) = Ent(D) - \sum_{v=1}^{V} \frac{|D^v|}{D} Ent(D^v)$$

假设我们对 32 支球队一无所知，现在我们分别添加两条信息:1. 各支球队每次世界杯夺冠概率;2. 每支队伍在本年内各自获胜的概率。假设我们通过对相应数值的计算得到两个结果分别为 $Gain(D, 1) = 0.314159, Gain(D, 2) = 0.223456$，即信息 1 的增益比信息 2 要大。则我们可以知道:信息 1 对我们的用处更大，它更能帮我们判断哪支队伍能获得冠军，因此在面对不得不删除两条信息中的一条时，我们可以删除第二条以达到最大程度上保留正确判断的结果。

还记得我们的目的吗？猜出谁是冠军，即比赛已经分出胜负，我们只需要通过信息来不断缩小目标范围即可。在西瓜问题上，我们通过不断加入约束条件以判断瓜的好坏，图9很好地说明了这一点:

此外，银行的放贷系统也可以通过决策树来判断，假设 A 银行请求我们为其做一个自动判断是否给予贷款的自动判别系统，他们提供了此前 15 组申请贷款的相应信息如图10，请你设计系统策略:

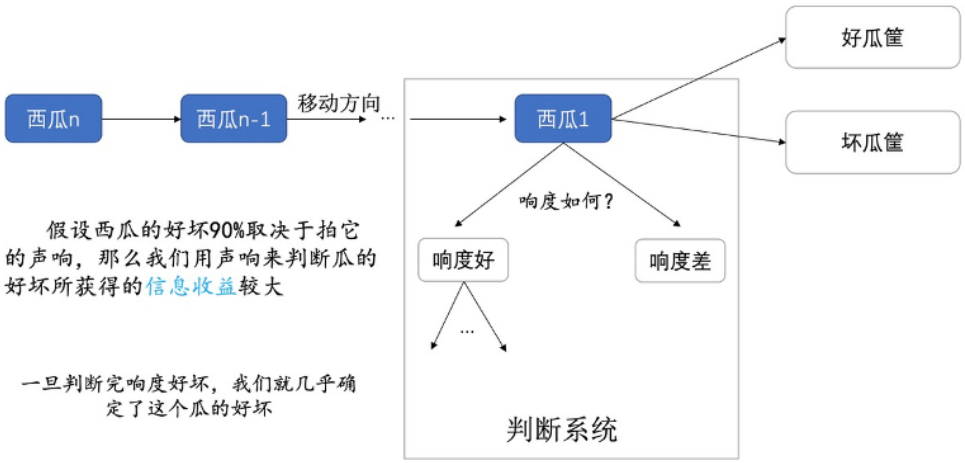


图 9: 判断西瓜好坏标准选择

值得注意的是，机器学习是通过预测未出现的情况并给出决策判断，我们设计的目的也并不是为了能够处理和之前情况相同的机器，那样做将毫无意义。事实上，刚刚的“猜队伍”比喻并不是十分恰当，信息熵的公式本质是概率密度分布函数，它是我们所需求信息相对于整体的分布概率。在银行是否借贷问题上，我们判断一个属性会产生多大的信息增益，可以通过该属性的样本所占样本空间比例进行计算，例如上表中我们可以分别计算青年、中年与老年所占的比例进而求得年龄这一维度信息的信息增益大小。我们分别计算每个属性 (年龄、有工作、有自己的房子等等) 的信息增益，并通过比较它们的大小来决定谁对是否能申请贷款影响最大。

经过上述模型的处理，你已经能够判断什么人获得贷款的几率最大。然而，此时的系统仍然只是一个无情的判断机器，它并不能预测与学习。换句话说，此时系统判断一个人能否获得贷款的决策顺序是十分固定的，例如他可能先判断这个人年龄，再判断信誉情况等等。这仍然是一个较为死板的系统，即对于一个问题，我只能设计出一种固定模式的系统，这样很不好，我们将之称为**欠拟合**。

有没有解决办法呢？有，换别的算法。

上述方法的策略无非是通过加入某个信息，看看系统中的信息增益如何。既然问题出在“概率密度函数”上，正向已经无法求得预期结果，那么我们反向思考：直接从结果中看不一致的概率。下面定义**基尼值**：

$$Gini(D) = \sum_{k=1}^{|y|} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{|y|} p_k^2$$

属性 a 的基尼指数定义为：$Gini\_index(D,a) = \sum_{v=1}^{V} \frac{|D^v|}{D} Gini(D^v)$，即作出决策判断的最小属性作为最优化分：$a* = argmin Gini\_index(D,a)$.

如何理解基尼指数? 客观来说，基尼值反映了数据集 D 中随机抽取两个样本，其类别标记不一致的概率。基尼值越小，数据集 D 的纯度越高。

如何理解类别标记不一致的概率? 举个例子: 某银行获得贷款名单中男女各半，那么你是否会觉得成功贷款与性别有关呢? 很显然，此时的性别选项中不一致概率较大 (男女概率各为 50 %)，基尼值较大，性别与是否成功贷款没有直接关系; 再考虑一个例子，如果银行获得贷款的人群都是

贷款申请样本数据表

ID	年龄	有工作	有自己的房子	信贷情况	·类别
1	青年	否	否	一般	否
2	青年	否	否	好	否
3	青年	是	否	好	是
4	青年	是	是	一般	是
5	青年	否	否	一般	否
6	中年	否	否	一般	否
7	中年	否	否	好	否
8	中年	是	是	好	是
9	中年	否	是	非常好	是
10	中年	否	是	非常好	是
11	老年	否	是	非常好	是
12	老年	否	是	好	是
13	老年	是	否	好	是
14	老年	是	否	非常好	是
15	老年	否	否	一般	否

图 10: 判断是否贷款的评判标准

20-30 岁的年轻人，即任意两个获得贷款的人在年龄上不一致的概率几乎为 0，那么你是否觉得年龄和贷款成不成功有关系呢? 此时，不一致概率极低，基尼值小，该属性的选择作用较好，年龄与是否贷款关系非常大，甚至年龄可能是作为判断能否贷款的唯一条件!

我们在把目光放到世界杯球队猜冠军问题与西瓜好坏问题。对于前者而言，若把一条信息加入判断条件后，我们若不能在其中找出两个帮助我们进一步判断的信息，那么这个信息无疑是无用的，即各个球队在该信息下不一致概率极大 (巴西、德国等等三十二支球队在这个信息加入后仍然在判断序列之中)，基尼值较大，即该信息与判断谁是冠军没有任何关系，我们仍然需要 5 个 bit 判断谁是冠军。对于西瓜问题，假设我们加入了西瓜产地这一信息，但是加入后好瓜坏瓜仍然都在数据集中，即加入了这一信息后瓜好坏不一致概率极大，即基尼值极大，数据无法区分，那么我们可知产地和西瓜好坏没有直接的联系。

为了保证生成的树拥有一定的预测能力，我们不对每一属性都进行决策。即不能对决策树干预过多，要让它有自己的发挥空间，因此需要对其剪枝处理。通常有两种策略:

1. 预剪枝: 生成树时对节点进行评估，若不能带来足够收益则禁止其生长.

2. 后剪枝: 生成树后，评估每个节点的收益，收益不好则删除.

此外，西瓜的各个属性并不一定都能用汉语的某个词汇进行描述，例如甜度，它是一个从 0-1 分布的连续值，我们的决策树也需要对这些属性进行处理。通常会采用二分法与重复划分等等，如下:

在实际操作中，有些西瓜不愿意提供自己的“敲响”这一属性，因为它们觉得这是自己的个人 (瓜) 隐私，如何处理这些缺失值? 其实解决方法也十分简单，既然人家不愿意提供自己的隐私信息，那么我们就不再强求。在计算信息增益时，直接以所有非缺失值之和作为分母进行计算即可，缺失值直接当做不存在处理。

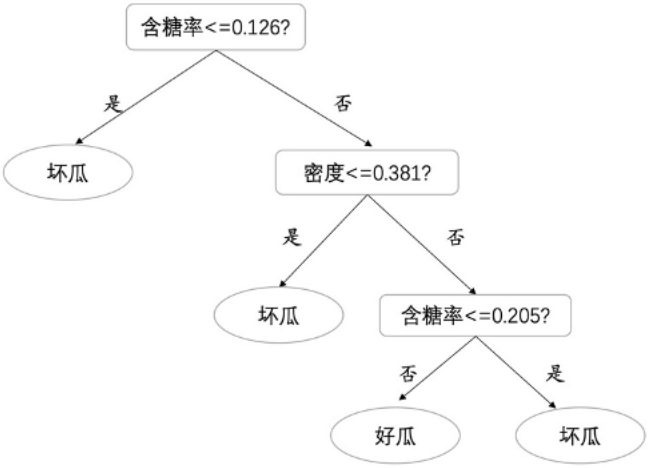


图 11: 决策模型

6.3 决策树代码

sklearn 中是有决策树的 API 的，其代码如下：

```
class sklearn.tree.DecisionTreeClassifier
(criterion='gini',max_depth=None,rando m_state=None)
#决策树分类器
#criterion:默认是'gini'系数，也可以选择信息增益熵的'entropy'
#max_depth:树的深度大小
#radom_state:随机数种子
#decision_path:返回决策树的路径
```

练习任务：

泰坦尼克号生存问题：通过泰坦尼克号上的乘客的个人信息，利用决策树来预测他 (她) 们最终是否存活。例如: 对女性和儿童而言，她们的生存概率更大，因此我们可以将性别、年龄纳入考虑范围等等。

课前准备：

1. 在 Terminal 中安装 graphviz.
2. 所需数据路径:http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt.

参考代码：

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction import DictVectorizer
from sklearn.tree import DecisionTreeClassifier, export_graphviz

def decision():
    #导入csv文件，选取x为特征值，包括社会阶级、年龄、性别，并确定我们的目标值，判断是否存活
    titan = pd.read_csv("TitanicDataSets.csv")
```

```
9       x = titan[['pclass','age','sex']]
10      y = titan['survived']
11      print(x)
12      #缺失值处理
13      x['age'].fillna(x['age'].mean(),inplace=True)
14
15      #分割一部分数据到测试集里
16      x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
17
18      #特征处理,特征值抽成字典格式
19      dict = DictVectorizer(sparse=False)
20      x_train = dict.fit_transform(x_train.to_dict(orient="records"))
21      print(dict.get_feature_names())
22      x_test = dict.transform(x_test.to_dict(orient="records"))
23      #print(x_train)
24
25      #决策树登场
26      dec = DecisionTreeClassifier()
27      dec.fit(x_train, y_train)
28
29      #预测准确率
30      print("预测准确率:", dec.score(x_test, y_test))
31
32      #导出决策树
33      export_graphviz(dec, out_file="./tree.dot",
34      feature_names=['年龄 ','pclass=1st','pclass=2nd','pclass=3rd','女性','男性'])
35
36      return None
37
38  decision()
39  #转换 dot 至 png, 在 terminal 中:
40  #1.cd ~/Downloads/pythonDemo/
41  #2.dot -Tpng tree.dot -o tree.png
```

6.4 随机森林

单一的决策树学习能力和处理问题的能力始终是有限的，在对上面的代码测试过程中，我们每运行一次就会生成一棵新的决策树。可不可以同时运行多个决策树，然后让系统选出最优的一棵呢？答案是肯定的，我们将这种思想称为**随机森林 (Random Forest)**，是集成学习的一种。

我们始终坚信：没有最好的分类器，只有最适合的分类器。随机森林平均来说最强，但也只在 9.9% 的数据集上拿到了第一；SVM 的平均水平紧随其后，在 10.7% 的数据集上拿到第一。神经网络 (13.2%) 和 boosting(~ 9%) 表现不错。数据维度越高，随机森林就比 AdaBoost 强越多，但是整体不及 SVM。数据量越大，神经网络就越强。

所谓的集成学习，就是指建立几个模型组合来解决单一的预测问题。它的工作原理是生成多个分类器/模型，各自独立地学习和作出预测。这些预测最后相互结合，最终输出一个预测，因此优于任何一个单独预测的结果。假设你训练了 5 个树，其中有 4 个树的结果是 True，1 个树的结果是 False，那么最终结果会是 True。

在随机森林过程中，我们采取随机抽样 (bootstrap)，并对样本进行放回再抽取。若不对结果放回处理，那么每一棵树都需要取走整体数据集中的一个部分，那么每一棵树都不是普适的，它都有自己的“喜好”，这样做是十分不利于整体模型的建立，因此采取随机放回抽取处理。

6.5 随机森林代码

sklearn 中是有随机森林的 API 的，其代码如下：

```
class sklearn.ensemble.RandomForestClassifier
(n_estimators=10,criterion='gini',max_depth=None, bootstrap=True, random_state=None)

#随机森林分类器
#n_estimators=integer,optional(default=10)森林里的树木数量：120，200，300，500棵
#criterion:string，可选(default="gini")分割特征的测量方法
#max_depth:integer或None，可选树的最大深度5，8，15，25，30等
#max_features="auto"，每个决策树的最大特征数量
#If"auto", then 'max_features=sqrt(n_features)'.
#If"sqrt", then 'max_features=sqrt(n_features)'(same as "auto").
#If"log2", then 'max_features=log2(n_features)'.
#If None then 'max_features=n_features'.
#bootstrap:Boolean, optional(default = True)构建树时是否采取放回重新抽样策略
```

练习任务：

利用随机森林来预测泰坦尼克号乘客是否存活（数据集同上）。

参考代码：

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction import DictVectorizer
from sklearn.ensemble import RandomForestClassifier

def decision():
    #导入csv文件，选取x为特征值，包括社会阶级、年龄、性别，并确定我们的目标值，判断是否存活
    titan = pd.read_csv("TitanicDataSets.csv")
    x = titan[['pclass','age','sex']]
    y = titan['survived']
    print(x)
    #缺失值处理
    x['age'].fillna(x['age'].mean(),inplace=True)

    #分割一部分数据到测试集里
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)

    #特征处理,特征值抽成字典格式
    dict = DictVectorizer(sparse=False)
    x_train = dict.fit_transform(x_train.to_dict(orient="records"))
    x_test = dict.transform(x_test.to_dict(orient="records"))

    #随机森林进行预测(超参数调优)
    rf = RandomForestClassifier()
    param = {"n_estimators":[120,200,300,500,800,1200],"max_depth":[5,8,15,25,30]}

    #网格搜索与交叉验证
    gc = GridSearchCV(rf,param_grid=param, cv=2)
    gc.fit(x_train, y_train)
    print("准确率：",gc.score(x_test,y_test))
    print("查看选择的参数模型:",gc.best_params_)
```

```
32
33      return None
34
35  decision()
```

6.6 线性回归

对于西瓜而言，我们若把“色泽”、“根蒂”、“敲声”作为三个坐标轴，则它们张成了一个用于描述西瓜的三维空间，每个西瓜都可以在这个空间中找到自己的坐标位置。由于空间中的每一个点对应一个坐标向量，因此我们把一个示例 (“色泽”、“根蒂”和“敲声”) 称为一个“特征向量”。在代数中，一个坐标系在经过线性变换（矩阵变换）后，方向不变的向量方称之为特征向量，这二者之间有什么联系吗?

事实上，在代数中我们通常策略是通过线性变换（即矩阵）来使向量更便于表示。而不管权重如何分配（扩大或缩小 λ 倍），“色泽”、“根蒂”和“敲声”始终可以作为描述西瓜属性的一组基。换种说法，我们可以写出：$h(w) = w_0 + w_1 \times$ 色泽 $+ w_2 \times$ 根蒂 $+ w_3 \times$ 敲声，其中 w_0, w_1, w_2, w_3 是不同属性的权重，即属性的重要性或重要程度。

为了总结出更一般性的规律，我们将上述公式整理一下，可得到：

$$h(w) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

记为 $h(w) = w^T x$，其中 w, x 为矩阵，$w = (w_0, w_1, w_2)^T, x = (1, x_1, x_2)^T$，前者为权重，后者为属性。

对于已有的一些西瓜，假设我们知道它们的所有属性并且知道其是不是一颗好瓜，如何才能设计一个系统让它自己能够根据样本从而辨别哪些是好瓜、哪些是坏瓜呢?

$$h(w) = w_0 + w_1 \times \text{色泽} + w_2 \times \text{根蒂} + w_3 \times \text{敲声}$$

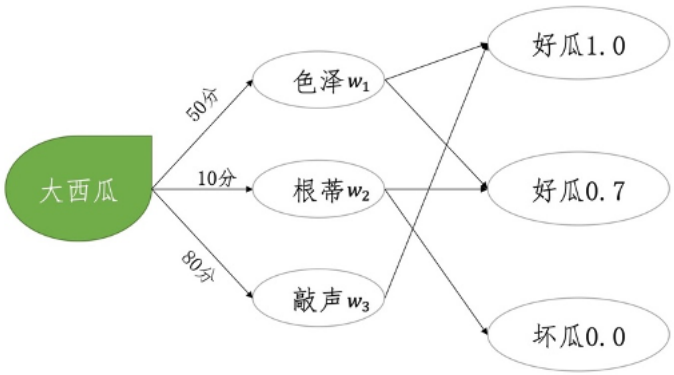


图 12: 好瓜的判断过程

事实上，西瓜的好坏是一个连续的值，世界并不是非黑即白的。我们通过给西瓜评分（0-1）来确定西瓜的具体情况，其中 1 代表最甜，0 代表最不好吃（其实做瓜做到这个份上也没有什么意思

了）。

如何得到反馈并调整我们的回归方程？设置损失函数：

$$I(\theta) = (h_w(x_1) - y_1)^2 + (h_w(x_2) - y_2)^2 + \cdots + (h_w(x_m) - y_m)^2 = \sum_{i=1}^{m}(h_w(x_i) - y_i)^2$$

这种方法又称为**最小二乘法 (OLS, Ordinary Least Squares)**。如何求解不同属性的权重值？即求不同的 w 成为了阻碍建立数学模型的最后一个问题。通常情况下我们有两种办法（以单变量为例）：

1. 正规方程（又称数学硬肝法）：直接求矩阵的解 $w = (X^T X)^{-1} X^T y$，其中 X 为特征矩阵，y 为目标矩阵.

2. 梯度下降（又称迭代求解法）：以单变量中的 w_0, w_1 为例，$w_1 = -w_1 - \alpha \frac{\partial cost(w_0 + w_1 x_1)}{\partial w_1}, w_0 = -w_0 - \alpha \frac{\partial cost(w_0 + w_1 x_1)}{\partial w_1}$，其中 α 为学习速率，需要手动指定，$\frac{\partial cost(w_0 + w_1 x_1)}{\partial w_1}$ 为梯度下降方向.

这似乎与我们平时学到的最小二乘法有一些不同。我们直接运用公式就可以求解，为什么要绕这么一大圈？变量较少的情况下，这种方法似乎都不如用公式手算得来的数据快，难道计算机算几个系数这么困难？

其实，这就是智能算法的核心所在。什么是智能算法？包括无人驾驶、智能机器人等等一系列的智能化设备，它们是如何运作的？智能算法不是无数的 if-else 嵌套，我们会面临无数种难以预料的情况。通常做法是：我们告诉机器哪些东西不能做（损失函数），哪些东西是我们要达到的目标（目标函数），这样让机器自己不断地学习、记忆，最终才能实现智能化。例如在设计无人车的自动避障算法，我们首先定义损失函数：越靠近障碍物，我们的损失函数要越大，这样才能让汽车避免撞击障碍物。不管汽车的初始位置如何，也不管障碍物在汽车的哪个方向，我们不在乎。这样让汽车的状态不断靠拢损失函数最小的情况，我们就实现了自动避障。

由此我们总结出一个一般性的思想：1. 随机设置初始值。2. 找到目标函数和损失函数。3. 令损失函数最小，目标函数最大，并令整个系统的状态不断向其靠拢。

回到最小二乘法问题上，常规的求解确实没有任何问题，但是如何让机器自动求解？我们没办法对每一种情况都写一个程序，这样太麻烦了，而且不现实。我们只需要告诉机器：我们的目标就是方差最小的直线方程（方差最小即为损失函数），每一步的循环过程和迭代方法我们都已经写好，至于如何求解，那是机器自己的事情了，该过程称之为**机器学习 (Machine Learning)**。

对于梯度下降，我们没法保证局部最优解就是全局最优解，我们需要用到另一些智能算法来解决这最关键的一个问题。事实上，智能算法正是为了解决这个问题而生的，例如神经网络、模拟退火和粒子群算法等等，尽管我们已经在整个思想和流程上达到了机器学习，但是算法并没有实现智能，具体的解决办法我们将在后面逐步介绍。

6.7 线性回归代码

练习任务：

波士顿房价问题：最小二乘法拟合各自变量与因变量波士顿房价之间的关系，并得出相应方程。

API 说明：

1. sklearn 中自带了 Boston 的数据集，因此你不需要额外下载 csv 文件。

2. sklearn.linear_model.LinearRegression：正规方程

3. sklearn.linear_model.SGDRegressor：梯度下降

参考代码：

```
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

def mylinear():
    #获取数据
    lb = load_boston()

    #分割数据集到训练集和测试集
    x_train, x_test, y_train, y_test = train_test_split(lb.data, lb.target,test_size=0.25)
    print(y_train,y_test)
    #进行标准化处理，目标值处理
    #特征值和目标值都必须进行标准化处理，实例两个标准化API
    std_x = StandardScaler()

    x_train = std_x.fit_transform(x_train)
    x_test = std_x.transform(x_test)

    #目标值
    std_y = StandardScaler()

    y_train = std_y.fit_transform(y_train.reshape(-1,1))
    y_test = std_y.transform(y_test.reshape(-1,1))

    #estimator预测
    #正规方程求解方式预测结果
    lr = LinearRegression()
    lr.fit(x_train, y_train)
    print(lr.coef_)

    #预测测试集房子价格
    y_lr_predict = std_y.inverse_transform(lr.predict(x_test))
    print("正规方程测试集里面每个房子的预测价格：",y_lr_predict)
    print("正规方程的均方误差:",mean_squared_error(std_y.inverse_transform(y_test),
                                                  y_lr_predict))

    #梯度下降法预测结果
    sgd = SGDRegressor()
    sgd.fit(x_train, y_train)
    print(sgd.coef_)

    #预测测试集的房子价格
    y_sgd_predict = std_y.inverse_transform((sgd.predict(x_test)))
    print("梯度下降法测试集里面每个房子的预测价格：",y_sgd_predict)
    print("梯度下降的均方误差：", mean_squared_error(std_y.inverse_transform(y_test),
                                                  y_lr_predict))

    return None

```

```
mylinear()
```

6.8 岭回归

统计学里最怕的一个问题就是**多重共线性**。线性回归只需要我们计算各个自变量的系数，而在科研中常常遇到的问题是：我们需要自己选择自变量种类。如何选择自变量？其中一个秘诀就是：独立不重复。如果我们选择的两个自变量种类相关性很强，甚至其中一个可以用另一个线性表示，那么显然这一次的结果意义不大，因为不能反映研究问题的统计特征。如果选择的变量之间相关性很强，我们就称结果存在着“多重共线性问题”。

事实上，**相关性**是一个很神奇的东西，有时候我们可能想不到两个变量之间会有什么联系，但是它们确实就存在着相关性。直观判断相关性并不可靠，我们有统计学的判断方法，在后面我们会进行讨论。

举个例子，假设我们在研究某产品的销量受价格、用户评分等的影响情况，建立了相应的线性回归方程如下：

$$Y = 5 + 2a - 3b + u$$

其中，a 为用户对产品的评分，b 为产品价格，因变量 Y 是销量，而 u 是其它可能存在的干扰项（产品的产地或其它我们没有考虑到的因素）。除了常数项 5 以外，方程的系数很好解释：其他不变的情况下，评分每增加一个单位，该产品的平均销量增加 2 个；价格每增加一个单位，产品平均销量减少 3 个。若评分为 0 且价格为 0 时，评分每增加一个单位，产品平均销量增加 5 个，这没有任何意义，因此我们不讨论常数项；u 是干扰项，包括其它我们没考虑周全的因素。

如果只是简单的直观感受，这完全没问题。但设想这样一种场景：该产品是通过互联网平台进行销售，销量在很大程度上也取决于网站的浏览量，那么这个模型还能说得通吗？如果网站的浏览量和用户对产品的评分相关性很强，并且我们又没有办法将网站浏览量单独提取成为一个自变量，我们就称此时的系统存在着极大的**内生性问题**，最后得到的结果差距一定巨大。

现在陷入了两难的处境：如果把浏览量作为自变量，那么存在着多重共线性问题；如果把浏览量作为扰动项，那么存在着内生性问题。权衡之下，我们还是选择把浏览量作为自变量，只不过我们要消除多重共线性的影响，方法就是**岭回归 (Ridge Regression)**。岭回归最早是用来处理特征数多于样本数的情况，现在也用来处理多重共线性问题，是一种有偏估计，其损失函数为：

$$J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2]$$

值得注意的是，岭回归的代价函数加入了一个 L2 正则项，没有正则项的是无偏估计，加入正则项的代价函数为有偏估计。

6.9 岭回归代码

sklearn 中是有岭回归的 API 的，其代码如下：

```
sklearn.linear_model.Ridge(alpha=1.0)
#具有 L2 正则化的线性最小二乘法
#alpha:正则化力度(0.0-1.0)
#coef_:回归系数
```

练习任务：

用岭回归计算波士顿房价问题。

```
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression, SGDRegressor,Ridge
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

def mylinear():
    #获取数据
    lb = load_boston()

    #分割数据集到训练集和测试集
    x_train, x_test, y_train, y_test = train_test_split(lb.data, lb.target,test_size=0.25)
    print(y_train,y_test)
    #进行标准化处理，目标值处理
    #特征值和目标值都必须进行标准化处理，实例两个标准化API
    std_x = StandardScaler()

    x_train = std_x.fit_transform(x_train)
    x_test = std_x.transform(x_test)

    #目标值
    std_y = StandardScaler()

    y_train = std_y.fit_transform(y_train.reshape(-1,1))
    y_test = std_y.transform(y_test.reshape(-1,1))

    #estimator预测
    #岭回归预测房价
    rd = Ridge(alpha=1.0)
    rd.fit(x_train, y_train)
    print(rd.coef_)

    #预测测试集的房子价格
    y_rd_predict = std_y.inverse_transform(rd.predict(x_test))
    print("岭回归测试集里每个房子的预测价格:",y_rd_predict)
    print("岭回归均方误差:",mean_squared_error(std_y.inverse_transform(y_test),y_rd_predict))

    return None

mylinear()
```

6.10 逻辑回归

假设我们有大量的癌症病人病例，上面标注了他们身体的各种医学信息，例如细胞的形态，细胞膜的状态等等。假设这些因素都与癌症的形成密切相关，我们能否写出一种算法根据这些变量信息预测一个人是否得病？答案是肯定的，得病与否我们可以看成回归的结果是 1 还是 0，这样答案就只有两种可能：得病与未得病。我们将这种非此即彼的情况称为**逻辑回归 (Logistic Regression)**。

逻辑回归是回归吗？这一点毫无疑问，此外，这种非此即彼的情况不也正是一种分类吗？逻辑回归也可以看成简单二分类的一种。逻辑回归是如何实现的？事实上，我们仅仅需要对线性回归进行一些小改动：我们让结果连续地映射到 [0,1] 区间，并设置一个阈值（例如 0.5），大于该阈值的

令其为 1，否则为 0，Sigmoid 函数可以完成这一操作。

$$S(x) = \frac{1}{1+e^{-x}}$$

该函数求导较为容易，即：$S'(x) = S(x)(1 - S(x))$. 假设我们输入一串信息，通过随机生成的一串权重系数进行回归求解，然后通过 Sigmoid 得到最终结果为 0.82，我们如何理解这个值？这有两种解释方式：1. 我们预测该个体有 82% 的几率患病. 2. 如果我们预测 100 个相同信息的个体，将会有 18 个人会被错误确诊. 这个 82% 是准确的值吗？一定不是，因为我们是随机生成的权重系数进行求解的。那么该如何调整呢？首先我们要介绍一下**极大似然估计**.

假如有一个罐子，里面有黑白两种颜色的球，数目多少不知，两种颜色的比例也不知。我们想知道罐中白球和黑球的比例，但我们不能把罐中的球全部拿出来数。现在我们可以每次任意从已经摇匀的罐中拿一个球出来，记录球的颜色，然后把拿出来的球再放回罐中。这个过程可以重复，我们可以用记录的球的颜色来估计罐中黑白球的比例。假如在前面的一百次重复记录中，有七十次是白球，请问罐中白球所占的比例最有可能是多少？很多人马上就有答案了：70%。而其后的理论支撑是什么呢？

我们假设罐中白球的比例是 p，那么黑球的比例就是 1-p。因为每抽一个球出来，在记录颜色之后，我们把抽出的球放回了罐中并摇匀，所以每次抽出来的球的颜色服从同一独立分布。进行二项分布列式计算，得到带有 p 的公式，令它最大，就能求得 p 的概率，这就是**极大似然估计**。

回到刚才的问题上来，我们该怎么处理逻辑回归中的系数呢？我们将极大似然估计得出的概率 p 假设为正确答案，进而不断去迭代、调整系数，这样我们最终就能得到一个较为准确的逻辑回归系数。

在第 2 点的情况下设每一次判断正确的概率为 p，则我们可以通过二项分布得到概率值：$P(Right) = C_n^k p^k (1-p)^{n-k}$. 只需要令 $P(Right)$ 最大，就能求得 p 的值。将该值与预测出来的值比较，两者的差越小函数拟合的越好，不断迭代。值得注意的是，此处我们也同样利用梯度下降法优化。整理成一般思路，即为：

1. 随机设置 $y = b + a_1x_1 + a_2x_2 + \cdots + a_nx_n$ 的系数，然后将观测值带入通过 Sigmoid 函数映射到 [0,1] 区间，得到我们瞎猜的概率值 p_0.
2. 得到的概率值 p_0 这个值是否准确呢？我们与极大似然估计得到的概率 p 进行比较，要求的即为令 $|p - p_0|$ 最小的那一组系数.
3. 如何不断优化？通过梯度下降法进行迭代.

因此我们直接写出极大似然损失函数：

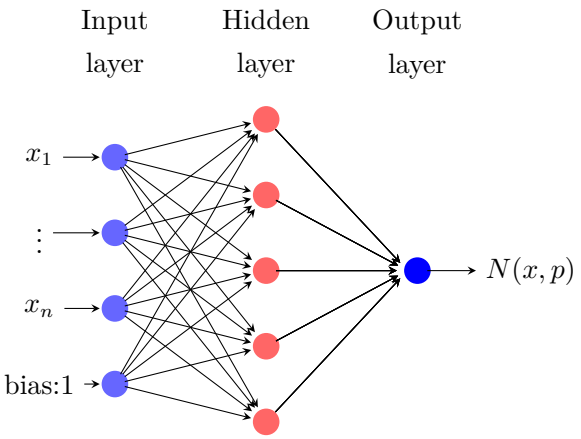
$$cost(h_\theta(x), y) = \sum_{i=1}^{m} -y_i log(h_\theta(x)) - (1-y_i) log(1 - h_\theta(x))$$

逻辑回归给我们提供了一种新的思路，由于最后可能的结果有个两个，因此我们采用交叉熵来描述整体的不确定程度，我们当然希望最终的不确定程度最小，也就是说我们算得的值和真实值之间差距越小，于是我们才引入极大似然的损失函数，也就是上文的交叉熵函数（又称二元熵，即为“非此即彼”的不确定程度），这对于我们有什么启示？我们完全可以把输出结果的 2 个，变为 3 个，4 个以至 n 个，于此，我们就得到了神经网络的雏形。

7 深度学习

7.1 神经网络

让我们来看一个经典的神经网络。这是一个包含三个层次的神经网络。它分别由输入层、输出层和中间层（也叫隐藏层）构成。



在开始介绍前，有一些知识可以先记在心里：

- 设计一个神经网络时，输入层与输出层的节点数往往是固定的，中间层则可以自由指定；
- 神经网络结构图中的拓扑与箭头代表着预测过程时数据的流向，跟训练时的数据流有一定的区别；
- 结构图里的关键不是圆圈（代表“神经元”），而是连接线（代表“神经元”之间的连接）。每个连接线对应一个不同的权重（其值称为权值），这是需要训练得到的。

神经网络的本质上是一种分类器，但不同于逻辑回归，它的输出种类可以有很多。在上文的西瓜例子中，我们对西瓜的各种参数进行输入并通过矩阵进行运算，如果我们不加激活函数，仅用矩阵进行变换的话，那么最终得到的结果一定是线性变换。那么是不是说神经网络只能不断逼近、拟合线性函数？对于非线性函数束手无策呢？

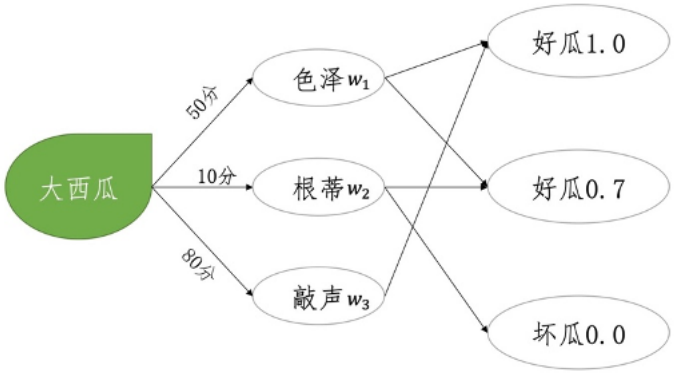


图 13: 好瓜的判断过程

如果我们在输入特征后，不对原数据进行矩阵变换，而是通过某些性质比较好的函数进行映射，由于函数并不都是线性的，因为函数的定义指出：函数的变换是包含了点对点的任意一种变换，这种变换可以是线性的，也可以是非线性的，并不是说：能写出表达式的函数就都是线性函数，这和矩阵有着较大区别。我们通过函数来代替矩阵对输入层进行“脱敏”处理，则该函数称为**“激活函数”**。

最初的感知器只能做简单的线性分类任务。但是当时的人们热情太过于高涨，并没有人清醒的认识到这点。于是，当人工智能领域的巨擘 Minsky 指出这点时，事态就发生了变化。

Minsky 在 1969 年出版了一本叫《Perceptron》的书，里面用详细的数学证明了感知器的弱点，尤其是感知器对 XOR（异或）这样的简单分类任务都无法解决。

Minsky 认为，如果将计算层增加到两层，计算量则过大，而且没有有效的学习算法。所以，他认为研究更深层的网络是没有价值的。由于 Minsky 的巨大影响力以及书中呈现的悲观态度，让很多学者和实验室纷纷放弃了神经网络的研究。神经网络的研究陷入了冰河期。这个时期又被称为“AI winter”。接近 10 年以后，对于两层神经网络的研究才带来神经网络的复苏。

两层神经网络是本文的重点，因为正是在这时候，神经网络开始了大范围的推广与使用。

Minsky 说过单层神经网络无法解决异或问题。但是当增加一个计算层以后，两层神经网络不仅可以解决异或问题，而且具有非常好的非线性分类效果。不过两层神经网络的计算是一个问题，没有一个较好的解法。1986 年，Rumelhar 和 Hinton 等人提出了反向传播（Backpropagation，BP）算法，解决了两层神经网络所需要的复杂计算量问题，从而带动了业界使用两层神经网络研究的热潮。目前，大量的教授神经网络的教材，都是重点介绍两层（带一个隐藏层）神经网络的内容。

这时候的 Hinton 还很年轻，30 年以后，正是他重新定义了神经网络，带来了神经网络复苏的又一春。

与单层神经网络不同。理论证明，两层神经网络可以无限逼近任意连续函数。这是什么意思呢？也就是说，面对复杂的非线性分类任务，两层（带一个隐藏层）神经网络可以分类的很好。也就是说，两层神经网络中，隐藏层对原始的数据进行了一个空间变换，使其可以被线性分类，然后输出层的决策分界划出了一个线性分类分界线，对其进行分类。

这样就导出了两层神经网络可以做非线性分类的关键–隐藏层。联想到我们一开始推导出的矩阵公式，我们知道，矩阵和向量相乘，本质上就是对向量的坐标空间进行一个变换。因此，隐藏层的参数矩阵的作用就是使得数据的原始坐标空间从线性不可分，转换成了线性可分。

两层神经网络通过两层的线性模型模拟了数据内真实的非线性函数。因此，多层的神经网络的本质就是复杂函数拟合。

下面来讨论一下隐藏层的节点数设计。在设计一个神经网络时，输入层的节点数需要与特征的维度匹配，输出层的节点数要与目标的维度匹配。而中间层的节点数，却是由设计者指定的。因此，“自由”把握在设计者的手中。但是，节点数设置的多少，却会影响到整个模型的效果。如何决定这个自由层的节点数呢？目前业界没有完善的理论来指导这个决策。一般是根据经验来设置。较好的方法就是预先设定几个可选值，通过切换这几个值来看整个模型的预测效果，选择效果最好的值作为最终选择。这种方法又叫做 **Grid Search（网格搜索）**。

如何选择激活函数？常见的激活函数有：Sigmoid、tanh、reLU 三种.

Sigmoid 函数表达式是：

$$f(x) = \frac{1}{1 + e^{-x}}$$

神经网络刚刚兴起时，使用 Sigmoid 作为激活函数十分常见，但是随着网络的不断发展，近年来已经很少使用，因为深层神经网络更新参数时，需要从输出层到输入层逐层进行链式求导，Sigmoid

函数的导数为 0-0.25 之间的小数，链式求导需要多层导数连续相乘，结果将趋于零导致梯度消失，使得参数无法进行更新；还有，我们希望经过激活函数的数值是以 0 为均值的，但经过 Sigmoid 函数不是，且输出的都是整数，会使收敛变慢；另外，Sigmoid 函数逆运算计算复杂度大，训练时间长。

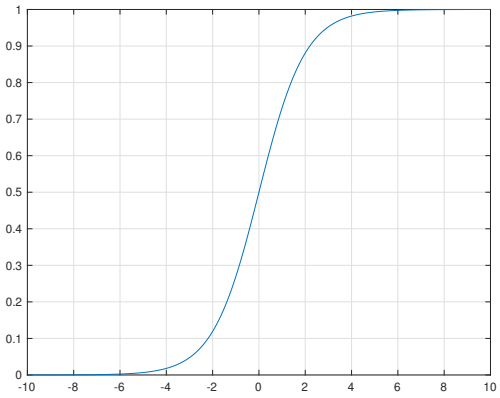


图 14: Sigmoid 函数

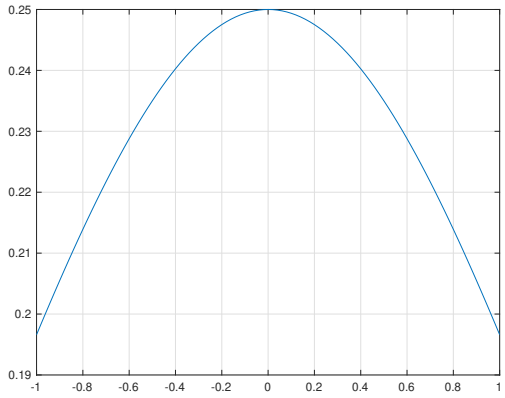


图 15: Sigmoid 的导函数

tanh 函数激活函数为：

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

从激活图像上看，tanh 函数的均值为 0 均值，但是依然存在梯度消失和逆运算复杂的问题。

reLU 的函数为：

$$f(x) = max(0, x) = \begin{cases} x & if x > 0 \\ 0 & if x \leqslant 0 \end{cases}$$

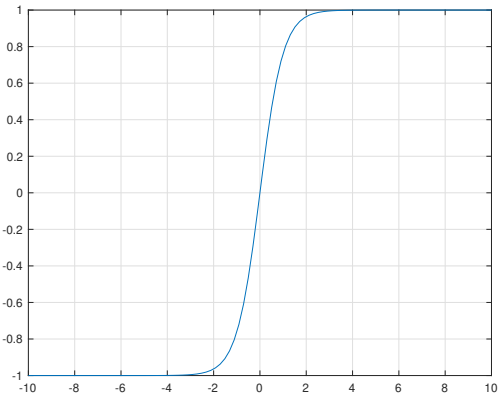


图 16: tanh 函数

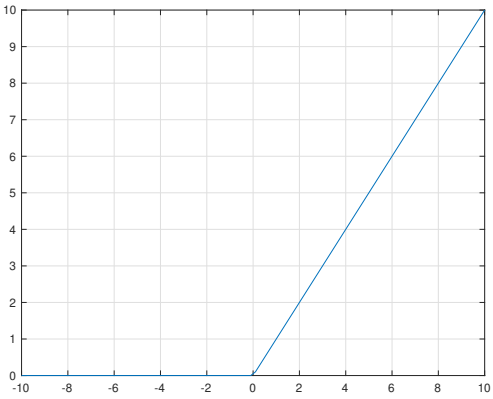


图 17: reLU 的函数

如果不用激活函数，每一层输出都是上层输入的线性函数，无论神经网络有多少层，输出都是输入的线性组合，这种情况就是最原始的**感知机（Perceptron）**。如果使用的话，激活函数给神经元引入了非线性因素，使得神经网络可以任意逼近任何非线性函数，这样神经网络就可以应用到众多的非线性模型中。

反向传播算法的启示是数学中的链式法则。在此需要说明的是，尽管早期神经网络的研究人员努力从生物学中得到启发，但从 BP 算法开始，研究者们更多地从数学上寻求问题的最优解。不再盲目模拟人脑网络是神经网络研究走向成熟的标志。正如科学家们可以从鸟类的飞行中得到启发，但没有必要一定要完全模拟鸟类的飞行方式，也能制造可以飞天的飞机。

优化问题只是训练中的一个部分。机器学习问题之所以称为学习问题，而不是优化问题，就是因为它不仅要求数据在训练集上求得一个较小的误差，在测试集上也要表现好。因为模型最终是要部署到没有见过训练数据的真实场景。提升模型在测试集上的预测效果的主题叫做**泛化（generalization）**，相关方法被称作**正则化（regularization）**。神经网络中常用的泛化技术有**权重衰减**等。

两层神经网络在多个地方的应用说明了其效用与价值。10 年前困扰神经网络界的异或问题被轻松解决。神经网络在这个时候，已经可以发力于语音识别，图像识别，自动驾驶等多个领域。

历史总是惊人的相似，神经网络的学者们再次登上了《纽约时报》的专访。人们认为神经网络可以解决许多问题。就连娱乐界都开始受到了影响，当年的《终结者》电影中的阿诺都赶时髦地说一句：我的 CPU 是一个神经网络处理器，一个会学习的计算机。

但是神经网络仍然存在若干的问题：尽管使用了 BP 算法，一次神经网络的训练仍然耗时太久，而且困扰训练优化的一个问题就是局部最优解问题，这使得神经网络的优化较为困难。同时，隐藏层的节点数需要调参，这使得使用不太方便，工程和研究人员对此多有抱怨。

90 年代中期，由 Vapnik 等人发明的 **SVM（Support Vector Machines，支持向量机）**算法诞生，很快就在若干个方面体现出了对比神经网络的优势：无需调参；高效；全局最优解。基于以上种种理由，SVM 迅速打败了神经网络算法成为主流。

回到上面的问题中，通过矩阵来连接每一层的函数所构成的这个感知机，在算法的数学原理上具有较高可解释性，本质上和前面的决策树、信息熵等等没有太大区别。仅当我们采用激活函数连接各个神经网络层时，我们才是第一次在算法上实现了“智能”。

7.2 Keras 搭建神经网络八股文

用 Tensorflow API：tf.keras 搭建网络八股六步法

1. **import**：导入 import tensorflow as tf 等相关模块
2. **train, test**：告知要喂入网络的训练集和测试集是什么，也就是要指定训练集的输入特征 x_train 和训练集的标签 y_train 分别是什么，还可以指定测试集的特征和标签
3. **model = tf.keras.models.Sequential**：第三层，在 Sequential() 中搭建网络结构，逐层描述每层网络，相当于走一遍前向传播
4. **model.compile**：在 compile() 中配置训练方法，告知选择哪种优化器、选择哪种损失函数、选择哪种评价指标
5. **model.fit**：在 fit() 中执行训练过程，告知训练集和测试集的输入特征和标签，告知每个 batch 是多少，告知要迭代多少次数据集

6. **model.summary**：用 summary() 打印出网络的结构和参数统计

7.2.1 Sequential 结构

Sequential 可以看作一个神经网络容器，逐层描述网络结构，每一层的网络结构可以是：

- 拉直层：tf.keras.layers.Flatten()
- 全连接层：tf.keras.layers.Dense(神经元个数，activation=" 激活函数"，kernel_regularizer= 哪种正则化)

  其中，activation（字符串给出）可选：relu、softmax、sigmoid、tanh
  kernel_regularizer 可选：tf.keras.regularizers.l1()、tf.keras.regularizers.l2()

- 卷积层：tf.keras.layers.Conv2D(filters = 卷积核个数，kernel_size = 卷积核，strides = 卷积步长，padding="valid" or "same")
- LSTM 层：tf.keras.layers.LSTM()

7.2.2 compile 结构

model.compile(optimizer = 优化器，loss = 损失函数, metrics = [" 准确率"])

```
//Optimizer可选:
'sgd' or tf.keras.optimizers.SGD(lr=学习率, momentum=动量参数)

'adagrad' or tf.keras.optimizers.Adagrad (lr=学习率)

'adadelta' or tf.keras.optimizers.Adadelta(lr=学习率)

'adam' or
tf.keras.optimizers.Adam(lr = 学习率, beta_1 = 0.9, beta_2=0.999)
```

```
//loss可选:
'mse' or tf.keras.losses.MeanSquaredError()

'sparse_categorical_crossentropy'  or
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False)
```

请注意，loss 函数中有一个 from_logits = False 选项，有些神经网络的输出是经过了 softmax 等函数的概率分布，有些则不经过概率分布直接输出，可以认为：该参数是在询问是否是原始输出，也就是没有经概率分布的输出，如果神经网络预测结果输出经过了概率分布，写入 False，否则为 True。很多情况下如果输出的神经网络准确率太差，很可能是这部分没有调节正确。

```
//Metrics可选:
'accuracy': y_和y都是数值, 如y_ = [1], y=[1]

'categorical_accuracy': y_和y都是独热码（概率分布）,
```

```
5  如y_ = [0, 1, 0], y=[0.256, 0.659, 0.048]
6
7  'sparse_categorical_accuracy': y_是数值，y是独热码（概率分布），
8  如y_ = [1], y=[0.256, 0.659, 0.048]
```

7.2.3 fit 结构

model.fit(训练集的输入特征，训练集的标签，
batch_size = , epochs = ,
validation_data = (测试集的输入特征，测试集的标签),
validation_split = 从训练集划分多少比例给测试集,
validation_freq = 多少次 epoch 测试一次)

7.3 iris 代码复现

鸢尾花，测量数据：花瓣的长度和宽度，花萼的长度和宽度，所有测量结果都以厘米为单位。这一类花有三个品种：setosa,versicolor,virginnica。我们的目标是建立一个基础的机器学习的模型预测鸢尾花的品种。sklearn 中内置了鸢尾花的数据集，我们只需要直接导入即可，在传统机器学习部分，我们一般会用 k 近邻算法来实现鸢尾花分类，此处我们采用神经网络进行分析：

```
1  import tensorflow as tf
2  from sklearn import datasets
3  import numpy as np
4
5  x_train = datasets.load_iris().data
6  y_train = datasets.load_iris().target
7
8  np.random.seed(116)
9  np.random.shuffle(x_train)
10 np.random.seed(116)
11 np.random.shuffle(y_train)
12 tf.random.set_seed(116)
13
14 model = tf.keras.models.Sequential([
15     tf.keras.layers.Dense(3, activation='softmax',
16     kernel_regularizer=tf.keras.regularizers.l2())
17 ])
18
19 model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),
20               loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
21               metrics=['sparse_categorical_accuracy'])
22
23 model.fit(x_train, y_train, batch_size=32,
24 epochs=500, validation_split=0.2, validation_freq=20)
25
26 model.summary()
```

7.4 训练 MNIST 数据集

当我们开始学习编程的时候，第一件事往往是学习打印”Hello World”。就好比编程入门有 Hello World，机器学习入门有 MNIST。MNIST 是一个入门级的计算机视觉数据集，它由全世界的爱好者们共同维护，来自世界各地的人们对上面的所有手写数字都进行了标签标记。学会训练让机器自动识别手写数字是深度学习的第一步，因此我们以该案例为切入点，开始学习 tensor 中的 keras：

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss=
              tf.keras.losses.SparseCategoricalCrossentropy
              (from_logits=False),
              metrics=['sparse_categorical_accuracy'])

model.fit(x_train, y_train, batch_size=32, epochs=5,
validation_data=(x_test, y_test), validation_freq=1)
model.summary()
```

7.5 断点续训与可视化

在我们训练 tensorflow 模型时，我们肯定不希望每运行一次代码就重新训练一次模型，正如在 sklearn 中我们保存模型一样，tensorflow 也支持断点续训，这样我们可以不断的迭代、训练我们的模型，此外，还可以通过加入可视化代码对训练的效果进行直观分析。

```
import tensorflow as tf
import os
import numpy as np
from matplotlib import pyplot as plt

np.set_printoptions(threshold=np.inf)

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
```

```
                loss=
                tf.keras.losses.SparseCategoricalCrossentropy
                (from_logits=False),
                metrics=['sparse_categorical_accuracy'])

checkpoint_save_path = "./checkpoint/mnist.ckpt"
if os.path.exists(checkpoint_save_path + '.index'):
    print('-------------load the model-----------------')
    model.load_weights(checkpoint_save_path)

cp_callback = tf.keras.callbacks.ModelCheckpoint
(filepath=checkpoint_save_path,
                            save_weights_only=True,
                            save_best_only=True)

history = model.fit(x_train, y_train, batch_size=32,
epochs=5, validation_data=(x_test, y_test), validation_freq=1,
                    callbacks=[cp_callback])
model.summary()

print(model.trainable_variables)
file = open('./weights.txt', 'w')
for v in model.trainable_variables:
    file.write(str(v.name) + '\n')
    file.write(str(v.shape) + '\n')
    file.write(str(v.numpy()) + '\n')
file.close()

############    show    ###############

# 显示训练集和验证集的acc和loss曲线
acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.subplot(1, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

7.6 卷积神经网络

卷积核:参数空间共享,卷积层提取空间信息。什么是卷积?卷积就是特征提取器,就是 CBAPD,而卷积神经网络就是指借助卷积核提取空间特征后，送入全连接网络。循环核具有记忆力，下图是 CNN 的整体过程。

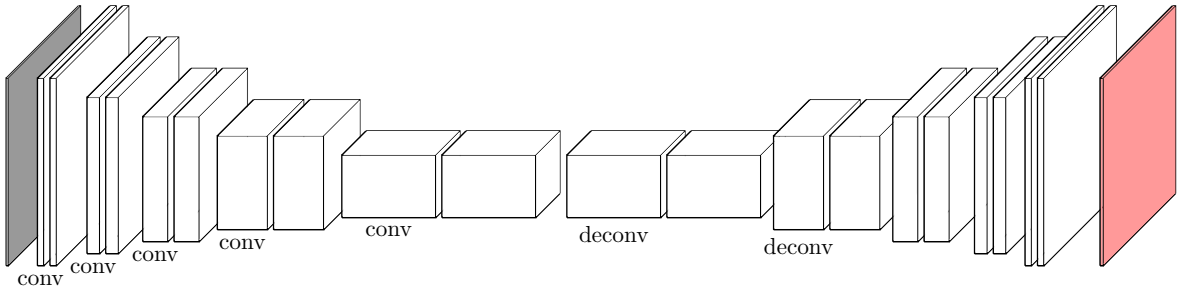
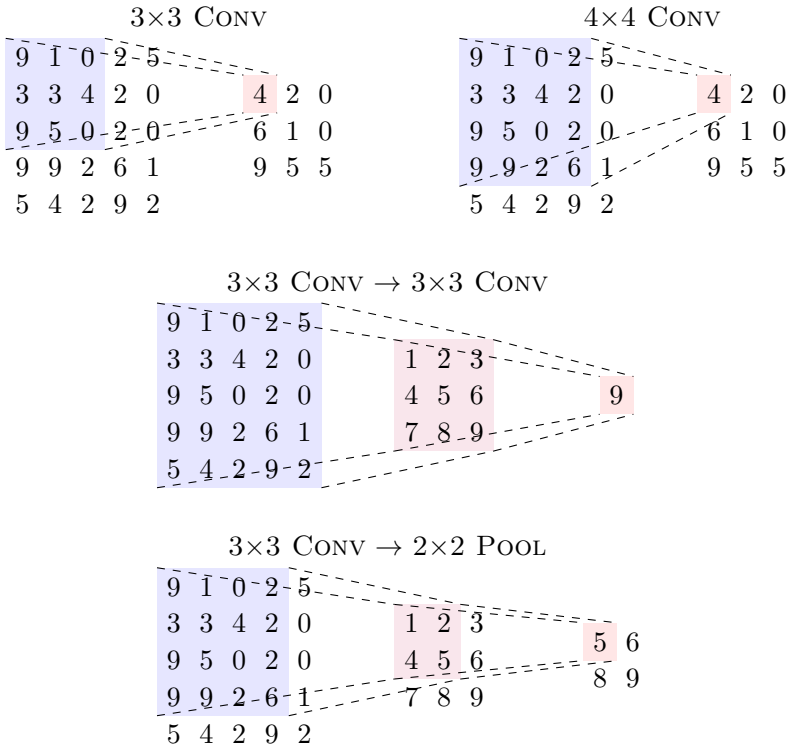


图 18: CNN 过程

事实上，CNN 是通过卷积核进行特征提取，其示意图如下

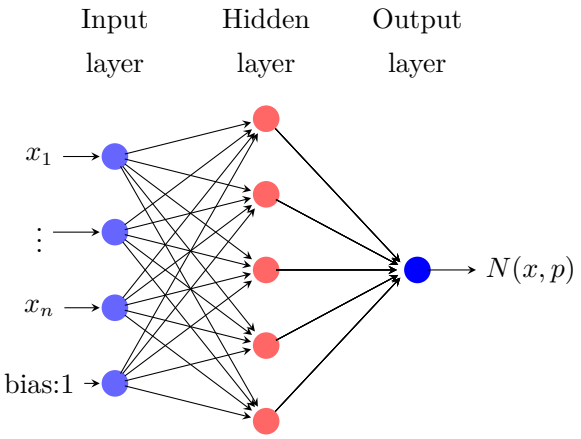


7.7 循环神经网络 RNN

我们常说矩阵的变换具有“线性”的特点，而线性又意味着条理性较强，因此我们把矩阵的“线性”看作是矩阵的“记忆性”。卷积神经网络通过提取一张图片的特征值，输把提取到的空间特征送入全连接网络中进行计算分析，然而，有些数据是与时间序列相关的，例如：“鱼离不开 _____”，我们的大脑中会自动提取“水”这个字作为答案，这种预测就是通过脑记忆体提取历史数据的特征，预测接下来可能发生的事。值得注意的是，时间序列数据往往意味着数据前后的相关性较强，而非一定要带上时间戳。循环核就是参数时间共享，循环层提取时间信息。

循环神经网络中，我们仍然用交叉熵来作为损失函数，整个过程也和全连接神经网络类似，不同的是，我们在激活函数上做一些处理。

全连接神经网络中，我们对激活函数的定义是：取代矩阵的、可以实现非线性映射的、连接两个计算层的桥梁。在此处，为了实现整个神经网络的记忆性，我们对激活函数之前的向量进行矩阵化处理，这样就能通过对矩阵参数的更新，来不断强化整个网络的记忆性，值得注意的是，我们在循环神经网络中对同一个功能所用的矩阵需要相同。听起来可能有些晦涩难懂，我们举个例子：



我们将三层神经网络分别用向量表示：

$$x_1 = \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \end{pmatrix}, \quad h_1 = \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{14} \end{pmatrix}, \quad y_1 = (y_1)$$

则我们首先对放入激活函数之前的向量进行矩阵处理，以此让它带有记忆性，即：$h_1 = tanh(W_{xh}x_1 + W_{hh}h_0 + b_1)$，其中 h_0 是基础项，也就是初始项，而 h 到 h 本身也存在矩阵变换，而 $y_1 = W_{hy} \cdot h_1 + b_2$，我们就完成了对原来激活函数的改进：

$$\text{激活函数} \longrightarrow \text{矩阵变换} + \text{激活函数}$$

值得注意的是，对于所有的输入变量 x，我们应用的都是同一个矩阵 W_{xh} 对其进行线性变换。如此，在应用过程中，我们就可以随机初始化矩阵，然后对结果进行比对，再不断修正矩阵来更新记忆参数了。

tf 提供了循环层计算的函数，tf.keras.layers.SimpleRNN(记忆体个数，activation=' 激活函数'. return_sequences = 是否每个时刻输出 ht 到下一层).

activation=' 激活函数'（不写，默认使用 tanh）

return_sequences=True 各时间步输出 ht

return_sequences=False 仅最后时间步输出 ht（默认），

例如：SimpleRNN(3, return_sequences=True)

下面我们将通过字母预测来进一步解释循环神经网络，例如：通过输入 a 预测出 b，输入 b 预测出 c，输入 c 预测出 d，输入 d 预测出 e，输入 e 预测出 a。首先我们要对五个字母用数字表示，最简单直接的方法就是独热码对五个字编码：a：10000，b：01000，c：00100，d：00010，e：00001

```
1  import numpy as np
```

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, SimpleRNN
import matplotlib.pyplot as plt
import os

input_word = "abcde"
w_to_id = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4}  # 单词映射到数值id的词典
id_to_onehot = {0: [1., 0., 0., 0., 0.], 1: [0., 1., 0., 0., 0.], 2: [0., 0., 1., 0., 0.], 3: [0., 0., 0., 1., 0.],
                4: [0., 0., 0., 0., 1.]}  # id编码为one-hot

x_train = [id_to_onehot[w_to_id['a']], id_to_onehot[w_to_id['b']], id_to_onehot[w_to_id['c']],
           id_to_onehot[w_to_id['d']], id_to_onehot[w_to_id['e']]]
y_train = [w_to_id['b'], w_to_id['c'], w_to_id['d'], w_to_id['e'], w_to_id['a']]

np.random.seed(7)
np.random.shuffle(x_train)
np.random.seed(7)
np.random.shuffle(y_train)
tf.random.set_seed(7)

# 使x_train符合SimpleRNN输入要求: [送入样本数, 循环核时间展开步数, 每个时间步输入特征个数]。
# 此处整个数据集送入，送入样本数为len(x_train)；输入1个字母出结果，循环核时间展开步数为1；表示为独热码有5个输入特征，每个时间步输入特征个数为5
x_train = np.reshape(x_train, (len(x_train), 1, 5))
y_train = np.array(y_train)

model = tf.keras.Sequential([
    SimpleRNN(3),
    Dense(5, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['sparse_categorical_accuracy'])

checkpoint_save_path = "./checkpoint/rnn_onehot_1pre1.ckpt"

if os.path.exists(checkpoint_save_path + '.index'):
    print('-------------load the model-----------------')
    model.load_weights(checkpoint_save_path)

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
                                                 save_weights_only=True,
                                                 save_best_only=True,
                                                 monitor='loss')
# 由于fit没有给出测试集，不计算测试集准确率，根据loss，保存最优模型

history = model.fit(x_train, y_train, batch_size=32, epochs=100, callbacks=[cp_callback])

model.summary()

# print(model.trainable_variables)
file = open('./weights.txt', 'w')  # 参数提取
```

```
for v in model.trainable_variables:
    file.write(str(v.name) + '\n')
    file.write(str(v.shape) + '\n')
    file.write(str(v.numpy()) + '\n')
file.close()

############    show   ###############

# 显示训练集和验证集的acc和loss曲线
acc = history.history['sparse_categorical_accuracy']
loss = history.history['loss']

plt.subplot(1, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.title('Training Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(loss, label='Training Loss')
plt.title('Training Loss')
plt.legend()
plt.show()

############### predict #############

preNum = int(input("input the number of test alphabet:"))
for i in range(preNum):
    alphabet1 = input("input test alphabet:")
    alphabet = [id_to_onehot[w_to_id[alphabet1]]]
    # 使alphabet符合SimpleRNN输入要求：[送入样本数，  循环核时间展开步数，  每个时间步输入特征个数]。此处验证效果送入了1个样本，
                                                    送入样本数为1；输入1个字母出结果，所以
                                                    循环核时间展开步数为1；表示为独热码有5
                                                    个输入特征，每个时间步输入特征个数为5
    alphabet = np.reshape(alphabet, (1, 1, 5))
    result = model.predict([alphabet])
    pred = tf.argmax(result, axis=1)
    pred = int(pred)
    tf.print(alphabet1 + '->' + input_word[pred])
```

7.8 LSTM

传统循环神经网络 RNN 可以通过记忆体实现短期记忆进行连续数据的预测，但是当连续数据的序列变长时，会使展开时间过长，在反向传播更新参数时，梯度要按照时间步连续相乘，会导致梯度消失。所以在 1997 年 Hochreitere 等人提出了长短记忆网络 LSTM。

LSTM 的数学原理较为简单，计算过程大致分为以下几个部分：

- 输入门（门限）：$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
- 遗忘门（门限）：$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
- 输出门（门限）：$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
- 细胞态（长期记忆）：$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

- 记忆体（短期记忆）：$h_t = o_t * tanh(C_t)$
- 候选态（归纳出的新知识）：$\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$

举个例子，LSTM 就是学习 tensorflow 的过程，现在你脑袋中记住的内容，是今天第 1 页 ppt 到第 45 页 ppt 的长期记忆 C_t，长期记忆 C_t 是由两部分组成，一部分是 ppt 第 1 页到 44 页的内容，也就是上一时刻的长期记忆 C_{t-1}。你不可能一字不差的记住全部内容，因此我们要乘一个遗忘门，这个乘积项表示留存在你脑中的对过去的记忆；现在我们学习的是新知识，是即将存在你脑中的记忆，现在部分由两部分组成：一部分是当前时刻的输入 x_t，还有一部分是第 44 页 ppt 短期记忆的留存，你脑袋把当前时刻的输入 x_t 和上一时刻的短期记忆 h_{t-1} 归纳形成即将存入你脑中的现在的记忆 $\tilde{C}_t$，现在的记忆乘以 $\tilde{C}_t$ 与过去的记忆一同构成你的长期记忆。

当你把所学到的知识复述给你的朋友时，你不可能一字不落的重复出来，你讲出来的是留存在你脑中的长期记忆，经过输出门筛选过后的内容，这就是记忆体的输出 h_t，当有多层循环网络时，第二层循环网络的输入 x_t，是第一层循环网络输出 h_t。输入第二层网络的是第一层网络提取出的精华，你可以认为：老师现在扮演的就是第一层循环网络，每一页 ppt 都是他们从文献中总结的精华，输出给我们。作为第二层循环网络的我们，接收到的数据，是老师的长期记忆经过 tanh 乘以输出门提取出的短期记忆 h_t，这就是 LSTM 的计算过程。如果我们再教给别人这些知识，那么还要经过第三层的运算。

如果你还没有掌握 LSTM 的计算过程，可以重新理解一下上面的例子，画一个思维导图；如果你自认为掌握的还不错，可以看一下 tensorflow 中给我们设定好的 LSTM 代码：

tf.keras.layers.LSTM(记忆体个数，return_sequences= 是否返回输出)

return_sequences=True 各时间步输出 ht

return_sequences=False 仅最后时间步输出 ht（默认）

例如：

```
model = tf.keras.Sequential([
  LSTM(80, return_sequences=True),
  Dropout(0,2),
  LSTM(100),
  Dropout(0,2),
  Dense(1)
])
```

下面给出 LSTM 预测股票的 tensorflow 代码：

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dropout, Dense, LSTM
import matplotlib.pyplot as plt
import os
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math

maotai = pd.read_csv('./SH600519.csv')  # 读取股票文件

```

```
training_set = maotai.iloc[0:2426 - 300, 2:3].values  # 前(2426-300=2126)天的开盘价作为训练集,表格从0开始计数, 2:3 是提取[2:3)列, 前闭后开,故提取出C列开盘价
test_set = maotai.iloc[2426 - 300:, 2:3].values  # 后300天的开盘价作为测试集

# 归一化
sc = MinMaxScaler(feature_range=(0, 1))  # 定义归一化: 归一化到(0, 1)之间
training_set_scaled = sc.fit_transform(training_set)  # 求得训练集的最大值, 最小值这些训练集固有的属性, 并在训练集上进行归一化
test_set = sc.transform(test_set)  # 利用训练集的属性对测试集进行归一化

x_train = []
y_train = []

x_test = []
y_test = []

# 测试集: csv表格中前2426-300=2126天数据
# 利用for循环, 遍历整个训练集, 提取训练集中连续60天的开盘价作为输入特征x_train, 第61天的数据作为标签, for循环共构建2426-300-60=2066组数据。
for i in range(60, len(training_set_scaled)):
    x_train.append(training_set_scaled[i - 60:i, 0])
    y_train.append(training_set_scaled[i, 0])
# 对训练集进行打乱
np.random.seed(7)
np.random.shuffle(x_train)
np.random.seed(7)
np.random.shuffle(y_train)
tf.random.set_seed(7)
# 将训练集由list格式变为array格式
x_train, y_train = np.array(x_train), np.array(y_train)

# 使x_train符合RNN输入要求: [送入样本数, 循环核时间展开步数, 每个时间步输入特征个数]。
# 此处整个数据集送入, 送入样本数为x_train.shape[0]即2066组数据; 输入60个开盘价, 预测出第61天的开盘价, 循环核时间展开步数为60; 每个时间步送入的特征是某一天的开盘价, 只有1个数据, 故每个时间步输入特征个数为1
x_train = np.reshape(x_train, (x_train.shape[0], 60, 1))
# 测试集: csv表格中后300天数据
# 利用for循环, 遍历整个测试集, 提取测试集中连续60天的开盘价作为输入特征x_train, 第61天的数据作为标签, for循环共构建300-60=240组数据。
for i in range(60, len(test_set)):
    x_test.append(test_set[i - 60:i, 0])
    y_test.append(test_set[i, 0])
# 测试集变array并reshape为符合RNN输入要求: [送入样本数, 循环核时间展开步数, 每个时间步输入特征个数]
x_test, y_test = np.array(x_test), np.array(y_test)
x_test = np.reshape(x_test, (x_test.shape[0], 60, 1))

model = tf.keras.Sequential([
    LSTM(80, return_sequences=True),
    Dropout(0.2),
    LSTM(100),
    Dropout(0.2),
    Dense(1)
])
```

```
60
61  model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
62                loss='mean_squared_error')  # 损失函数用均方误差
63  # 该应用只观测loss数值，不观测准确率，所以删去metrics选项，一会在每个epoch迭代显示时只显示
                                                    loss值
64
65  checkpoint_save_path = "./checkpoint/LSTM_stock.ckpt"
66
67  if os.path.exists(checkpoint_save_path + '.index'):
68      print('-------------load the model-----------------')
69      model.load_weights(checkpoint_save_path)
70
71  cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
72                                                   save_weights_only=True,
73                                                   save_best_only=True,
74                                                   monitor='val_loss')
75
76  history = model.fit(x_train, y_train, batch_size=64, epochs=50, validation_data=(x_test,
                                                    y_test), validation_freq=1,
77                      callbacks=[cp_callback])
78
79  model.summary()
80
81  file = open('./weights.txt', 'w')  # 参数提取
82  for v in model.trainable_variables:
83      file.write(str(v.name) + '\n')
84      file.write(str(v.shape) + '\n')
85      file.write(str(v.numpy()) + '\n')
86  file.close()
87
88  loss = history.history['loss']
89  val_loss = history.history['val_loss']
90
91  plt.plot(loss, label='Training Loss')
92  plt.plot(val_loss, label='Validation Loss')
93  plt.title('Training and Validation Loss')
94  plt.legend()
95  plt.show()
96
97  ################## predict ######################
98  # 测试集输入模型进行预测
99  predicted_stock_price = model.predict(x_test)
100 # 对预测数据还原---从（0，1）反归一化到原始范围
101 predicted_stock_price = sc.inverse_transform(predicted_stock_price)
102 # 对真实数据还原---从（0，1）反归一化到原始范围
103 real_stock_price = sc.inverse_transform(test_set[60:])
104 # 画出真实数据和预测数据的对比曲线
105 plt.plot(real_stock_price, color='red', label='MaoTai Stock Price')
106 plt.plot(predicted_stock_price, color='blue', label='Predicted MaoTai Stock Price')
107 plt.title('MaoTai Stock Price Prediction')
108 plt.xlabel('Time')
109 plt.ylabel('MaoTai Stock Price')
110 plt.legend()
111 plt.show()
112
113 ##########evaluate###############
114 # calculate MSE 均方误差 ---> E[(预测值-真实值)^2] (预测值减真实值求平方后求均值)
```

```
mse = mean_squared_error(predicted_stock_price, real_stock_price)
# calculate RMSE 均方根误差--->sqrt[MSE]    (对均方误差开方)
rmse = math.sqrt(mean_squared_error(predicted_stock_price, real_stock_price))
# calculate MAE 平均绝对误差----->E[|预测值-真实值|](预测值减真实值求绝对值后求均值)
mae = mean_absolute_error(predicted_stock_price, real_stock_price)
print('均方误差: %.6f' % mse)
print('均方根误差: %.6f' % rmse)
print('平均绝对误差: %.6f' % mae)
```

7.9 GRU

在 2014 年，cho 等人简化了 LSTM 结构，提出了 GRU 网络，GRU 使记忆体 h_t 融合了长期记忆和短期记忆，h_t 包含了过去信息 h_{t-1} 和现在信息 $\tilde{h}_t$，现在信息是过去信息过重置门与当前输入共同决定。两个门限的取值范围也是 0 到 1.

- 更新门：$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$
- 重置门：$r_t = \sigma(W_z \cdot [h_{t-1}, x_t])$
- 记忆体：$h_t = (1 - z_i) * h_{t-1} + z_t * \tilde{h}_t$
- 候选隐藏层：$\tilde{h}_t = tanh(W \cdot [r_t * h_{t-1}, x_t])$

前向传播时，直接使用记忆体更新公式，就能算出每个时刻的 h_t 的值

tf.keras.layers.GRU(记忆体个数，return_sequences= 是否返回输出)

return_sequences=True 各时间步输出 ht

return_sequences=False 仅最后时间步输出 ht（默认）

例如：

```
model = tf.keras.Sequential([
  GRU(80, return_sequences=True),
  Dropout(0,2),
  GRU(100),
  Dropout(0,2),
  Dense(1)
])
```

下面是运用 GRU 预测股票数据代码：

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dropout, Dense, GRU
import matplotlib.pyplot as plt
import os
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math

maotai = pd.read_csv('./SH600519.csv')  # 读取股票文件

```

```
training_set = maotai.iloc[0:2426 - 300, 2:3].values  # 前(2426-300=2126)天的开盘价作为训练集,表格从0开始计数, 2:3 是提取[2:3)列, 前闭后开,故提取出C列开盘价
test_set = maotai.iloc[2426 - 300:, 2:3].values  # 后300天的开盘价作为测试集

# 归一化
sc = MinMaxScaler(feature_range=(0, 1))  # 定义归一化: 归一化到(0, 1)之间
training_set_scaled = sc.fit_transform(training_set)  # 求得训练集的最大值, 最小值这些训练集固有的属性, 并在训练集上进行归一化
test_set = sc.transform(test_set)  # 利用训练集的属性对测试集进行归一化

x_train = []
y_train = []

x_test = []
y_test = []

# 测试集: csv表格中前2426-300=2126天数据
# 利用for循环, 遍历整个训练集, 提取训练集中连续60天的开盘价作为输入特征x_train, 第61天的数据作为标签, for循环共构建2426-300-60=2066组数据。
for i in range(60, len(training_set_scaled)):
    x_train.append(training_set_scaled[i - 60:i, 0])
    y_train.append(training_set_scaled[i, 0])
# 对训练集进行打乱
np.random.seed(7)
np.random.shuffle(x_train)
np.random.seed(7)
np.random.shuffle(y_train)
tf.random.set_seed(7)
# 将训练集由list格式变为array格式
x_train, y_train = np.array(x_train), np.array(y_train)

# 使x_train符合RNN输入要求: [送入样本数, 循环核时间展开步数, 每个时间步输入特征个数]。
# 此处整个数据集送入, 送入样本数为x_train.shape[0]即2066组数据; 输入60个开盘价, 预测出第61天的开盘价, 循环核时间展开步数为60; 每个时间步送入的特征是某一天的开盘价, 只有1个数据, 故每个时间步输入特征个数为1
x_train = np.reshape(x_train, (x_train.shape[0], 60, 1))
# 测试集: csv表格中后300天数据
# 利用for循环, 遍历整个测试集, 提取测试集中连续60天的开盘价作为输入特征x_train, 第61天的数据作为标签, for循环共构建300-60=240组数据。
for i in range(60, len(test_set)):
    x_test.append(test_set[i - 60:i, 0])
    y_test.append(test_set[i, 0])
# 测试集变array并reshape为符合RNN输入要求: [送入样本数, 循环核时间展开步数, 每个时间步输入特征个数]
x_test, y_test = np.array(x_test), np.array(y_test)
x_test = np.reshape(x_test, (x_test.shape[0], 60, 1))

model = tf.keras.Sequential([
    GRU(80, return_sequences=True),
    Dropout(0.2),
    GRU(100),
    Dropout(0.2),
    Dense(1)
])
```

```
60
61  model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
62                loss='mean_squared_error')  # 损失函数用均方误差
63  # 该应用只观测loss数值，不观测准确率，所以删去metrics选项，一会在每个epoch迭代显示时只显示
                                                    loss值
64
65  checkpoint_save_path = "./checkpoint/stock.ckpt"
66
67  if os.path.exists(checkpoint_save_path + '.index'):
68      print('-------------load the model-----------------')
69      model.load_weights(checkpoint_save_path)
70
71  cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
72                                                   save_weights_only=True,
73                                                   save_best_only=True,
74                                                   monitor='val_loss')
75
76  history = model.fit(x_train, y_train, batch_size=64, epochs=50, validation_data=(x_test,
                                                     y_test), validation_freq=1,
77                      callbacks=[cp_callback])
78
79  model.summary()
80
81  file = open('./weights.txt', 'w')  # 参数提取
82  for v in model.trainable_variables:
83      file.write(str(v.name) + '\n')
84      file.write(str(v.shape) + '\n')
85      file.write(str(v.numpy()) + '\n')
86  file.close()
87
88  loss = history.history['loss']
89  val_loss = history.history['val_loss']
90
91  plt.plot(loss, label='Training Loss')
92  plt.plot(val_loss, label='Validation Loss')
93  plt.title('Training and Validation Loss')
94  plt.legend()
95  plt.show()
96
97  ################## predict ######################
98  # 测试集输入模型进行预测
99  predicted_stock_price = model.predict(x_test)
100 # 对预测数据还原---从（0，1）反归一化到原始范围
101 predicted_stock_price = sc.inverse_transform(predicted_stock_price)
102 # 对真实数据还原---从（0，1）反归一化到原始范围
103 real_stock_price = sc.inverse_transform(test_set[60:])
104 # 画出真实数据和预测数据的对比曲线
105 plt.plot(real_stock_price, color='red', label='MaoTai Stock Price')
106 plt.plot(predicted_stock_price, color='blue', label='Predicted MaoTai Stock Price')
107 plt.title('MaoTai Stock Price Prediction')
108 plt.xlabel('Time')
109 plt.ylabel('MaoTai Stock Price')
110 plt.legend()
111 plt.show()
112
113 ##########evaluate###############
114 # calculate MSE 均方误差 ---> E[(预测值-真实值)^2] (预测值减真实值求平方后求均值)
```

```
115  mse = mean_squared_error(predicted_stock_price, real_stock_price)
116  # calculate RMSE 均方根误差--->sqrt[MSE]    (对均方误差开方)
117  rmse = math.sqrt(mean_squared_error(predicted_stock_price, real_stock_price))
118  # calculate MAE 平均绝对误差----->E[|预测值-真实值|](预测值减真实值求绝对值后求均值)
119  mae = mean_absolute_error(predicted_stock_price, real_stock_price)
120  print('均方误差: %.6f' % mse)
121  print('均方根误差: %.6f' % rmse)
122  print('平均绝对误差: %.6f' % mae)
```