Chapter 1

数据处理

1.1 数据清理终极指南

在过去的几个月里，我对来自传感器、调查以及日志的数据进行了分析。然而，无论我绘制了多少图表，运用了多么复杂的算法，结果总是存在误导性。

向数据运用随机森林算法，就如同向数据注入病毒一般。这是一种会无意中损害您的见解的“病毒”，仿佛您的数据在输出垃圾信息。

更糟糕的是，当您向首席执行官（CEO）展示您的新发现时，哎呀，您猜怎么着？他/她发现了一个缺陷，一些看起来不太对劲的地方，您的发现与他/她对该领域的理解不相符——毕竟，他/她是领域专家，比身为分析师或开发人员的您更了解情况。

刹那间，您的脸涨得通红，手也在颤抖，出现片刻的沉默，接着或许就是道歉。

但这还不是最糟糕的情况。如果您的发现被用作决策依据，而您的公司最终根据这些结果做出了决定，那又会怎样呢？

您摄取了一堆脏数据，却没有对其进行清理，然后您告知公司依据这些结果采取行动，结果却证明是错误的。那您可就有大麻烦了！

不正确或不一致的数据会导致错误的结论。因此，您对数据进行清理和理解的程度，对结果的质量有着重大影响。

维基百科上给出了两个真实的例子。

例如，政府可能希望分析人口普查数据，以便决定哪些地区在基础设施和服务方面需要进一步的支出与投资。在这种情况下，获取可靠的数据以避免做出错误的财政决策至关重要。

在商业领域，不正确的数据可能会付出高昂的代价。许多公司使用客户信息数据库来记录诸如联系信息、地址和偏好等数据。例如，如果地址不一致，公司将承担重新发送邮件甚至失去客户的成本。

正所谓“垃圾进，垃圾出”。

事实上，一个简单的算法可能会胜过一个复杂的算法，仅仅是因为它所使用的数据充足且质量高。

高质量的数据胜过花哨的算法。

例如，年龄（age）不能为负数，身高（height）同样不能为负数。其他规则可能涉及同一行中的多个列，或者跨数据集的多个列。

1.1.1 清洗

数据清理需要根据问题和数据类型采用不同的技术。可以应用多种方法，每种方法都有其自身的优缺点。

总体而言，不正确的数据要么被删除、更正，要么进行补充。

1.1.2 不相关的数据

不相关的数据是指那些实际上并不需要，且与我们试图解决的问题背景不相符的数据。

例如，如果我们要分析有关人口总体健康状况的数据，那么电话号码就不是必需的——从列的角度来看。

同样，如果您只对一个特定国家感兴趣，就不希望纳入所有其他国家的数据。或者，如果只研究那些去做手术的患者，我们就不会将所有人都包括在内——从行的角度来看。

只有当您确定某段数据不重要时，才能将其删除。否则，应探索特征变量之间的相关矩阵。

即便您没有发现任何相关性，也应该咨询领域专家。要知道，一个看似无关紧要的特征，从领域角度（比如临床角度）来看，可能实际上非常相关。

1.1.3 重复数据

重复项是指在数据集中重复出现的数据点。

它经常发生在例如：

- 数据来自不同的来源。
- 用户可能会点击两次提交按钮，以为表单实际上并未提交。
- 在线预订请求提交了两次，更正了第一次意外输入的错误信息。

常见的情况是两个用户具有相同的 ID 号。或者，同一篇文章被废弃了两次。

因此，这些重复项应该被删除。

1.1.4 类型转换

要确保数字存储为数字数据类型。日期应存储为日期对象或 Unix 时间戳（秒数）等。

如果有需要，可以将分类值转换为数字，或者从数字转换回分类值。

通过查看摘要中每列的数据类型，能够快速发现这一问题（我们在前面已经讨论过）。

需要注意的是，无法转换为指定类型的值应转换为 NA 值（或其他合适的值），并显示警告。这表明该值不正确，必须进行修复。

1.1.5 语法错误

删除空格

应删除字符串开头或结尾的多余空格。例如：`" hello world "` 应转换为 `"hello world"`。

填充字符串

字符串可以用空格或其他字符填充到一定宽度。例如，某些数字代码通常用前缀 0 填充，以确保它们始终具有相同的位数。例如：`313` 应转换为 `000313`（6 位数字）。

修复拼写错误

字符串可能会以多种不同的方式输入，出现错误也就不足为奇了。

例如，对于 `Gender` 这一分类变量：

m
Male
fem .
FemalE
Femle

此分类变量被认为具有 5 个不同的类别，而不是预期的 2 个类别：male 和 female，因为每个值都不一样。

可以使用条形图来可视化所有唯一值。人们可能会注意到一些值虽然不同，但实际上表示的是相同的事物，比如 `information_technology` 和 `IT`。或者，也许区别仅在于大小写，比如 `other` 和 `Other`。

因此，我们的任务是从上述数据中识别出每个值是男性还是女性。我们该如何做到这一点呢？

第一种解决方案是手动将每个值映射到 `male` 或 `female`。例如`:dataframe['gender'].map({'m':'male', 'fem.':'female',...})`

第二种解决方案是使用模式匹配。例如，我们可以在字符串开头的 `gender` 中查找 `m` 或 `M` 的出现。例如：`re.sub(`$r^{"}m$`", 'Male','male', flags=re.IGNORECASE)`

第三种解决方案是使用模糊匹配：一种识别预期字符串与每个给定字符串之间距离的算法。它的基本实现计算将一个字符串转换为另一个字符串需要多少次操作。

例如：

Gender	male	female
m	3	5
Male	1	3
fem.	5	3
FemalE	3	2
Femle	3	1

表 1.1: 模糊匹配示例结果

此外，如果您有一个像城市名称这样的变量，并且怀疑存在拼写错误或类似的字符串，也应该以同样的方式处理。例如，`lisbon` 可能被输入为 `lisboa`、`lisbona`、`Lisbon` 等。

City	Distance from ”lisbon”
lisbon	0
lisboa	1
Lisbon	1
lisbona	2
london	3
...	...

表 1.2: 城市名称距离示例

如果是这样，那么我们应该将所有表示相同内容的值替换为一个唯一值。在这种情况下，应将前 4 个字符串替换为 `lisbon`。

请注意 `0`、`不适用`、`NA`、`None`、`Null` 或 `INF` 等值，它们的含义可能相同：即缺少值。

1.1.6 规范

我们的职责不仅是识别拼写错误，还要将每个值都采用相同的标准化格式。

对于字符串，要确保所有值均为小写或大写。

对于数值，要确保所有值都具有特定的度量单位。

例如，高度可以用米和厘米作为单位。1 米的差异应被视为与 1 厘米的差异相同。因此，这里的任务是将高度转换为一个统一的单位。

对于日期，美国版本与欧洲版本有所不同。将日期记录为时间戳（毫秒数）与将日期记录为 date 对象也不一样。

1.1.7 扩展 / 转换

缩放是指对数据进行转换，使其适合特定的比例，例如 0 - 100 或 0 - 1。

例如，学生的考试成绩可以重新缩放为百分比（0 - 100），而不是绩点（0 - 5）。

它还有助于使某些类型的数据更易于绘制。例如，我们可能希望减少偏度以帮助绘图（当存在大量异常值时）。最常用的函数有对数（log）、平方根（square root）和倒数（inverse）。

还可以对具有不同度量单位的数据进行缩放。

例如，学生在不同考试中的分数，比如 SAT 和 ACT，由于这两项考试的评分等级不同，无法直接进行比较。1 个 SAT 分数的差异应被视为与 1 个 ACT 分数的差异相同。在这种情况下，我们需要重新调整 SAT 和 ACT 分数，使其取值范围在 0 - 1 之间。

1.2 决策树

本文是关于决策分析中的决策树的。有关该术语在机器学习中的用法，请参阅决策树学习。

传统上，决策树是手动创建的。决策树是一种决策支持递归分区结构，它运用树状模型来分析决策及其可能产生的后果，包括偶然事件结果、资源成本和效用。这是展示仅包含条件控制语句的算法的一种方式。决策树通常用于运筹学，尤其是决策分析，以帮助确定最有可能实现目标的策略，同时它也是机器学习中的常用工具。

1.2.1 概述

决策树是一种类似流程图的结构，其中每个内部节点代表对一个属性的“测试”（例如，抛硬币是正面还是反面），每个分支代表测试的结果，每个叶节点代表一个类别标签（在计算所有属性后做出的决策）。从根节点到叶节点的路径表示一条分类规则。

在决策分析中，决策树和密切相关的影响图被用作可视化和分析决策的支持工具，其中会计算竞争替代方案的期望值（或预期效用）。

决策树由三种类型的节点组成：

1. 决策节点——通常用正方形表示
2. 机会节点——通常用圆圈表示
3. 结束节点——通常用三角形表示

决策树通常用于运筹学和运营管理。如果在实际情况中，必须在信息不完全知晓的情况下在线做出决策，并且没有历史数据可供参考，那么应将决策树与概率模型作为最佳选择模型或在线选择模型算法并行使用。决策树的另一个用途是作为计算条件概率的描述性手段。

决策树、影响图、效用函数以及其他决策分析工具和方法，由商学院、卫生经济学和公共卫生学院的本科生学习，它们是运筹学或管理科学方法的示例。这些工具还用于预测户主在正常和紧急情况下的决策。

1.2.2 决策树构建块

决策树元素

从左到右绘制，决策树只有分支节点（拆分路径），但没有汇聚节点（收敛路径）。因此，手动绘制它们可能会变得非常庞大，而且通常很难完全靠手工绘制出来。传统上，决策树是手动创建的——正如旁边的例子所示——尽管现在越来越多地使用专门的软件。

决策规则

决策树可以线性化为决策规则，其中结果是叶节点的内容，沿路径的条件在 `if` 子句中形成一个合取式。通常，规则的形式为：如果条件 1 且条件 2 且条件 3，则结果。可以通过构建与右侧目标变量相关的关联规则来生成决策规则。它们还可以表示时间或因果关系。

使用流程图符号的决策树

通常，使用流程图符号绘制决策树，因为这样更容易被许多人阅读和理解。请注意，如下所示的树的“继续（Proceed）”计算中存在概念错误；该错误与法律诉讼中判给的“费用”的计算有关。

1.2.3 分析示例

分析可以考虑决策者（例如公司）的偏好或效用函数，例如：

在这种情况下，基本解释是，在现实风险偏好系数（大于 $400,000——处于风险厌恶范围内，公司需要模拟第三种策略，“既不是 A 也不是 B”）下 B 的风险和收益。

另一个通常用于运筹学课程的例子是海滩上救生员的分配（又名“海滩人生（Life's a Beach）”示例）。该示例描述了两个海滩，每个海滩上都分配了救生员。最大预算 B 可以在两个海滩之间分配（总共），并且使用边际回报表，分析师可以决定为每个海滩分配多少救生员。

每个海滩上的救生员	总共防止了溺水事件，海滩 #1	总共防止了溺水事件，海滩 #2
1	3	1
2	0	4

表 1.3: 救生员分配示例数据

在这个例子中，可以绘制一个决策树来说明海滩 #1 的收益递减原则。

决策树表明，在按顺序分配救生员时，如果只有 1 名救生员的预算，则在海滩 #1 上放置一名救生员将是最佳选择。但是，如果有两名救生员的预算，那么将两名救生员都放在海滩 #2 可以防止更多的整体溺水事件。

1.2.4 影响图

决策树中的许多信息可以更紧凑地表示为影响图，将注意力集中在问题和事件之间的关系上。

左侧的矩形表示决策，椭圆表示操作，菱形表示结果。

1.2.5 关联规则归纳

决策树也可以看作是从经验数据中归纳规则的生成模型。然后将最佳决策树定义为能够涵盖大部分数据，同时最大限度地减少层级（或“问题”）数量的树。已经设计了几种算法来生成这种最优树，例如 ID3/4/5、CLS、CART 等。

1.2.6 优点和缺点

优点

在决策支持工具中，决策树（和影响图）具有以下优点：

- 易于理解和解释。经过简短的解释后，人们就能够理解决策树模型。
- 即使几乎没有确凿的数据也有价值。可以根据专家对情况（其替代方案、概率和成本）及其对结果的偏好的描述，生成重要的见解。
- 有助于确定不同场景的最差值、最佳值和期望值。
- 使用白盒模型。如果给定的结果由模型提供，可以追溯其决策过程。
- 可以与其他决策技术结合使用。
- 可以考虑多个决策者的行动。

缺点

决策树的缺点：

- 它们不稳定，这意味着数据的微小变化可能会导致最佳决策树的结构发生巨大变化。
- 它们通常相对不准确。许多其他预测变量在处理类似数据时表现更好。这可以通过将单个决策树替换为决策树的随机森林来补救，但随机森林并不像单个决策树那样容易解释。
- 对于包含具有不同级别数量的分类变量的数据，决策树中的信息增益偏向于那些具有更多级别的属性。
- 计算可能会变得非常复杂，尤其是在许多值不确定和/或许多结果相互关联的情况下。

1.2.7 优化决策树

在提高决策树分类器的准确性时，应考虑一些事项。以下是在确保生成的决策树模型做出正确的决策或分类时要考虑的一些可能的优化方法。请注意，这些并不是唯一需要考虑的事情，而只是其中一部分。

增加树的层级数

决策树的准确性可能会根据决策树的深度而变化。在许多情况下，树的叶节点是纯节点。当一个节点是纯节点时，这意味着该节点中的所有数据都属于一个类别。例如，如果数据集中的类别是癌症和非癌症，那么当叶节点中的所有样本数据仅属于一个类别（癌症或非癌症）时，叶节点将被视为纯节点。需要特别注意的是，在优化决策树时，更深的树并不总是更好。更深的树可能会对运行时产生负面影响。如果正在使用某种分类算法，那么更深的树可能意味着此分类算法的运行时间明显变慢。此外，随着树的深入，构建决策树的实际算法也有可能明显变慢。如果使用的树构建算法拆分纯节点，则树分类器的整体准确性可能会降低。有时，深入树通常会导致准确性降低，因此测试修改决策树的深度并选择产生最佳结果的深度非常重要。总而言之，观察以下几点，我们将数字 D 定义为树的深度。

增加数字 D 的可能好处：

- 决策树分类模型的准确性提高。

增加 D 的可能缺点：

- 运行时问题
- 总体准确性下降
- 纯节点拆分在深入时可能会导致问题。

在更改 D 时测试分类结果差异的能力是必不可少的。我们必须能够轻松地更改和测试可能影响决策树模型准确性和可靠性的变量。

节点拆分函数的选择

使用的节点拆分函数可能会对提高决策树的准确性产生影响。例如，使用信息增益函数可能比使用 phi 函数产生更好的结果。phi 函数被称为衡量在决策树中的某个节点处拆分的候选者“优劣”的指标。信息增益函数被称为“熵减少”的量度。在下文中，我们将构建两个决策树。将使用 phi 函数构建一个决策树来拆分节点，并使用信息增益函数构建一个决策树来拆分节点。

信息增益和 phi 函数的主要优缺点：

- 信息增益的一个主要缺点是，被选为树中下一个节点的特征往往具有更多的唯一值。

- 信息增益的一个优点是，它倾向于选择最接近树根的最具影响力的特征。这是决定某些特征相关性的非常好的衡量标准。

- phi 函数也是根据“优劣”确定某些特征相关性的一个很好的度量。

信息增益函数公式：

$$IG(D,t) = H(D) - \sum_{i \in \{left,right\}} \frac{|D_i|}{|D|} H(D_i)$$

其中，$IG(D,t)$ 表示信息增益，$H(D)$ 是决策树节点的熵，D_i 是节点 t 处候选分裂后的子集，$|D|$ 和 $|D_i|$ 分别是相应集合的大小。

phi 函数公式：

$$\phi(D,t) = \sum_{i \in \{left,right\}} \frac{|D_i|}{|D|} \left(1 - \sum_{j=1}^{k} p_{ij}^2\right)$$

当所选特征以产生同质分割的方式分割样本并且每次分割中的样本数量大致相同时，ϕ 函数将最大化，其中 p_{ij} 是子集 D_i 中类别 j 的比例，k 是类别总数。

我们将 D（我们正在构建的决策树的深度）设置为 3（$D = 3$）。我们还有以下癌症和非癌症样本的数据集，以及样本具有或不具有的突变特征。如果样本具有特征突变，则样本对该突变为正，并且它将由 1 表示。如果样本没有特征突变，则该样本对于该突变为负，它将用零表示。

总而言之，C 代表癌症，NC 代表非癌症。字母 M 代表突变，如果样本具有特定的突变，它将在表中显示为 1，否则为 0。

M1 系列	M2 系列	M3 系列	M4 系列	M5 系列	类别
0	1	0	1	1	C1
0	0	0	0	0	NC1
0	0	1	1	0	NC2
0	0	0	0	0	NC3
1	1	1	1	1	C2
0	0	0	1	0	NC4

表 1.4: 示例数据集

现在，我们可以使用这些公式来计算数据集中每个 M 的 ϕ 函数值和信息增益值。计算出所有值后，就可以生成树。首先要做的是选择根节点。在信息增益和 ϕ 函数中，我们认为最佳分裂是产生最高信息增益或 ϕ 函数值的突变。现在假设 $M1$ 具有最高的 ϕ 函数值，$M4$ 具有最高的信息增益值。$M1$ 突变将是 ϕ 函数树的根，$M4$ 将是信息增益树的根。

现在，一旦我们选择了根节点，我们就可以根据样本的根节点突变是阳性还是阴性，将样本分成两组。这些组将称为组 A 和组 B。例如，如果我们使用 $M1$ 来拆分根节点中的样本，我们会得到 A 组中的 $NC2$ 和 $C2$ 样本，B 组中的其余样本 $NC4$、$NC3$、$NC1$、$C1$。

忽略为根节点选择的突变，继续将具有最高信息增益或 ϕ 函数值的下一个最佳特征放置在决策树的左侧或右侧子节点中。一旦我们为 $depth = 3$ 的树选择了根节点和两个子节点，我们就可以添加叶子了。叶子将代表模型根据样本具有或没有的突变生成的最终分类决策。左树是我们通过使用信息增益来拆分节点得到的决策树，右树是我们通过使用 ϕ 函数来拆分节点得到的。

现在假设使用混淆矩阵给出两个树的分类结果。

信息增益混淆矩阵：

ϕ 函数混淆矩阵：

	预测：癌症	预测：非癌症
实际：癌症	1	1
实际：非癌症	0	4

表 1.5: 信息增益混淆矩阵

	预测：癌症	预测：非癌症
实际：癌症	2	0
实际：非癌症	1	3

表 1.6: ϕ 函数混淆矩阵

使用信息增益的树在计算准确性时与使用 ϕ 函数的树具有相同的结果。当我们根据使用信息增益的模型对样本进行分类时，我们得到一个真阳性、一个假阳性、零个假阴性和四个真阴性。对于使用 ϕ 函数的模型，我们得到两个真阳性、零个假阳性、一个假阴性和三个真阴性。下一步是使用一些关键指标评估决策树的有效性，这些指标将在下面的评估决策树部分中讨论。下面将讨论的指标可以帮助确定在优化决策树时应采取的下一步措施。

其他技术

上述信息并不是构建和优化决策树的全部内容。有许多技术可以改进我们构建的决策树分类模型。其中一种技术是从自助抽样数据集构建我们的决策树模型。自助抽样数据集有助于消除在使用与测试模型相同的数据构建决策树模型时出现的偏差。利用随机森林的能力也可以显著提高所构建模型的整体准确性。这种方法从许多决策树中生成许多决策，并统计每个决策树的投票以做出最终分类。有许多技术，但主要目标是尝试以不同的方式构建决策树模型，以确保其达到尽可能高的性能水平。

1.2.8 评估决策树

了解用于评估决策树的指标很重要。主要使用的指标包括准确性、敏感度、特异性、精确度、漏诊率、错误发现率和错误遗漏率。所有这些指标都是从通过决策树分类模型运行一组样本时获得的真阳性、假阳性、真阴性和假阴性的数量推导出来的。此外，可以制作一个混淆矩阵来显示这些结果。所有这些主要指标都反映了基于决策树构建的分类模型的优缺点。例如，低敏感度和高特异性可能表明从决策树构建的分类模型在识别癌症样本与非癌症样本方面表现不佳。

让我们看一下下面的混淆矩阵。混淆矩阵显示我们构建的决策树模型分类器给出了 11 个真阳性、1 个假阳性、45 个假阴性和 105 个真阴性。

	预测：癌症	预测：非癌症
实际：癌症	11	45
实际：非癌症	1	105

表 1.7: 示例混淆矩阵

我们现在将计算准确性、敏感度、特异性、精确度、漏诊率、错误发现率和错误遗漏率的值。

准确性：

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

敏感度（真阳性率，TPR）：

$$Sensitivity = \frac{TP}{TP + FN}$$

特异性（真阴性率，TNR）：

$$Specificity = \frac{TN}{TN + FP}$$

精确度（阳性预测值，PPV）：

$$Precision = \frac{TP}{TP + FP}$$

漏诊率（假阴性率，FNR）：

$$FNR = \frac{FN}{TP + FN}$$

错误发现率（FDR）：

$$FDR = \frac{FP}{TP + FP}$$

错误遗漏率（FOR）：

$$FOR = \frac{FN}{TN + FN}$$

一旦我们计算出关键指标，我们就可以对构建的决策树模型的绩效做出一些初步结论。我们计算出的准确性为 71.60%。这个准确性值作为起点还不错，但我们希望在保持整体性能的同时，使我们的模型尽可能准确。敏感度值为 19.64%，这意味着在所有实际患有癌症的人中，检测为阳性的比例。如果我们看特异性值为 99.06%，我们可知在所有癌症检测为阴性的样本中，实际为阴性的比例。在敏感度和特异性方面，在这两个值之间取得平衡很重要，因此如果我们能够降低特异性以提高敏感度，那将是有益的。这些只是一些关于如何使用这些值以及它们背后的含义来评估决策树模型并在下一次迭代中进行改进的示例。

1.3 SIFT（Scale-Invariant Feature Transform）算法详解

SIFT 代表 Scale-Invariant Feature Transform，于 2004 年由不列颠哥伦比亚大学的 D.Lowe 首次提出。SIFT 具有图像缩放和旋转不变性的特点。需要注意的是，这个算法是有专利的，所以它被包含在 OpenCV 的 Non-free 模块中。

1.3.1 SIFT 的主要优点

1. **Locality（局部性）**：特征是局部的，因此对遮挡和杂波具有很强的鲁棒性（无需事先进行图像分割）。
2. **独特性**：单个特征能够与大型对象数据库进行匹配。
3. **数量**：即使是较小的物体也可以生成众多特征。
4. **效率**：具备接近实时的性能表现。
5. **可扩展性**：可以轻松扩展到各种不同的功能类型，每种类型都有助于增强其稳健性。

SIFT 是 7 系列特征检测和匹配的一部分，除此之外还有其他相关文章对其进行阐述。

1.3.2 算法

SIFT 是一种较为复杂的算法，主要涉及四个步骤，下面我们将逐一进行了解。

1. **尺度空间峰选择**：用于查找特征的潜在位置。
2. **关键点本地化**：准确定位特征关键点。
3. **方向分配**：为关键点指定方向。
4. **关键点描述符**：将关键点描述为高维向量。

1.3.3 关键点匹配

尺度空间峰值选择

Scale-space（缩放空间） 现实世界中的对象仅在特定比例下才有意义。例如，您可以在桌子上清晰地看到方糖，但如果从观察整个银河系的角度来看，它就几乎可以忽略不计了。对象的这种多尺度特性在自然界中十分常见，而 scale space（尺度空间）就是试图在数字图像上模拟这一概念。

图像的缩放空间是一个函数 $L(x, y, \sigma)$，它是由不同比例的高斯核（模糊）与输入图像的卷积产生的。Scale-space（尺度空间）被划分成多个八度音程，八度音程的数量以及音阶取决于原始图像的大小。由此，我们可以生成原始图像的若干个八度音程，并且每个八度音程的图像大小是前一个八度音程图像大小的一半。

模糊 在一个八度音程内，使用 Gaussian Blur（高斯模糊）运算符逐渐模糊图像。从数学角度来讲，“模糊”就是高斯算子和图像的卷积运算。高斯模糊针对每个像素有着特定的表达式或“运算符”，经过运算后会得到模糊的图像。

用数学表达式表示为：$G(x, y, \sigma) * I(x, y)$，其中 G 是 Gaussian Blur（高斯模糊）运算符，I 是图像，x、y 是位置坐标，σ 是“scale”（尺度）参数，可将其理解为模糊量，其值越大，图像模糊程度越高。

DOG（高斯核差） 现在，我们利用这些模糊的图像来生成另一组图像，即高斯差分（DoG）图像。这些 DoG 图像非常有助于在图像中找出有价值的关键点。高斯差分是通过两个具有不同 σ 值（设为 σ 和 $k\sigma$）的图像的高斯模糊差值得到的，此过程针对高斯金字塔中图像的不同八度音程来完成，如下图所示：

查找关键点

到目前为止，我们已经生成了一个尺度空间，并利用该尺度空间计算出了高斯差分。然后，这些结果被用于计算尺度不变的高斯拉普拉斯近似。

具体做法是将图像中的 1 个像素与其 8 个相邻像素以及下一个比例中的 9 个像素和上一个比例中的 9 个像素进行比较，这样总共要进行 26 次检查。如果该像素是一个局部极值，那么它就是一个潜在的关键点，这意味着该关键点在当前尺度下得到了最佳体现。

关键点本地化

上一步生成的关键点数量较多，其中部分关键点位于图像边缘，或者它们的对比度不够，在这两种情况下，这些关键点的作用相对有限。所以我们需要去除这些不理想的关键点，该方法与 Harris Corner Detector（哈里斯角点检测）中用于移除边缘特征的方法类似。对于低对比度特征，只需检查它们的强度即可。

具体而言，通过使用尺度空间的泰勒级数展开来获取更精确的极值位置，如果该极值的强度小于阈值（依据相关论文，阈值设定为 0.03），则将其舍弃。由于 DoG 对边缘具有较高的响应，所以还需要去除边缘特征，这里使用 2×2 的 Hessian 矩阵（H）来计算主曲率。

方向分配

现在我们得到了合格的关键点，它们已经过检测，是比较稳定的，并且我们也知道了检测到这些关键点的尺度（与模糊图像的尺度相同），所以已经具备了尺度不变性。接下来要做的是为每个关键点分配一个方向，以此实现旋转不变性。

围绕关键点位置根据其尺度选取一个邻域，然后在该区域内计算梯度幅值和方向。创建一个具有 36 个区间、覆盖 360 度的方向直方图。例如，若某点（在“方向收集区域”内）的梯度方向是 18.759 度，那么它就会被归入 10 - 19 度这个区间。并且添加到该区间的“量”与该点的梯度幅值成正比。对关键点周围的所有像素都完成上述操作后，直方图会在某个位置出现峰值。

选取直方图中的最高峰，同时将高于最高峰 80% 的其他峰值也考虑进来，用于计算方向。这样就创建出了位置和尺度相同，但方向不同的关键点，有助于提高匹配的稳定性。

关键点描述符

此时，每个关键点都有了位置、尺度和方向信息，接下来要为每个关键点周围的局部图像区域计算一个描述符，使其尽可能具有独特性，并且对诸如视角和光照变化等因素具有不变性。

为此，选取关键点周围的一个 16×16 的窗口，将其划分为 16 个 4×4 大小的子块。

针对每个子块，创建一个具有 8 个区间的方向直方图。

所以在实际应用中，对于 16×16 的样本阵列使用了 4×4 个描述符，$4 \times 4 \times 8$ 个方向可得出 128 个区间值，将其表示为一个特征向量，以此形成关键点描述符。不过这个特征向量会带来一些复杂问题，在最终确定特征向量之前需要解决这些问题。

旋转依赖性 特征向量使用了梯度方向，显然，如果旋转图像，所有的梯度方向都会改变。为实现旋转独立性，需要将每个方向减去关键点自身的旋转角度，这样每个梯度方向都是相对于关键点的方向而言的。

光照依赖性 如果对较大的数值设置阈值，就可以实现光照独立性。具体来说，将特征向量中的任何大于 0.2 的数值都改为 0.2，然后对得到的特征向量再次进行归一化处理，这样就得到了一个光照独立的特征向量。

关键点匹配

通过识别两个图像中关键点的最近邻来进行关键点匹配。但在某些情况下，第二近的匹配点可能与最近的匹配点距离非常接近，这种情况可能是由于噪声或其他原因导致的。针对这种情况，计算最近距离与次近距离的比值，如果该比值大于 0.8，则将其舍弃。根据相关论文所述，这样做可以消除大约 90% 的错误匹配，同时仅舍弃 5% 的正确匹配。

1.4 贪心算法的原理与应用

贪心算法是一种在每一步都做出当前看起来最优选择的算法策略。它旨在通过一系列的局部最优决策来达到全局最优解，然而，这并非总是能够成功。

贪心算法通常用于解决那些具有明显阶段性决策，且每个阶段的最优选择相对容易确定的问题。其核心原理在于，在每一个决策点，都依据某种既定的标准选择当前看来最优的选项，而不考虑该选择对后续步骤的影响。

例如，在找零钱的问题中，我们的目标是用最少的硬币组合找零给定的金额。贪心算法会在每一步都选择面值最大且不超过剩余金额的硬币。这种策略在这个特定问题中能够有效地找到最优解，因为硬币的面值是固定且有序的。

1.5 克鲁斯卡尔算法

克鲁斯卡尔算法是贪心算法在求解最小生成树问题上的典型应用。以使用最少的电线来连接几个网点为例，假设我们有若干个网点，以及连接这些网点的可能线路及其对应的成本（权值）。

以下是一个算法示例：

```
# 定义一个边的类
class Edge:
def __init__(self, src, dest, weight):
self.src = src
self.dest = dest
self.weight = weight

# 定义一个并查集的类
class UnionFind:
def __init__(self, n):
self.parent = [i for i in range(n)]
self.rank = [0 for _ in range(n)]

def find(self, i):
if self.parent[i] == i:
return i
self.parent[i] = self.find(self.parent[i])
return self.parent[i]

def union(self, x, y):
x_root = self.find(x)
y_root = self.find(y)

if self.rank[x_root] < self.rank[y_root]:
self.parent[x_root] = y_root
elif self.rank[x_root] > self.rank[y_root]:
self.parent[y_root] = x_root
```

```
else:
self.parent[y_root] = x_root
self.rank[x_root] += 1

# 克鲁斯卡尔算法函数
def kruskal(edges, num_vertices):
uf = UnionFind(num_vertices)
sorted_edges = sorted(edges, key=lambda x: x.weight)
mst = []
for edge in sorted_edges:
src_root = uf.find(edge.src)
dest_root = uf.find(edge.dest)
if src_root!= dest_root:
mst.append(edge)
uf.union(src_root, dest_root)
return mst

# 示例输入
edges = [
Edge(0, 1, 2),
Edge(0, 3, 6),
Edge(1, 2, 3),
Edge(1, 3, 8),
Edge(1, 4, 5),
Edge(2, 4, 7),
Edge(3, 4, 9)
]
num_vertices = 5

# 运行克鲁斯卡尔算法并输出结果
minimum_spanning_tree = kruskal(edges, num_vertices)
for edge in minimum_spanning_tree:
print(f"连接网点 {edge.src} 和 {edge.dest}, 成本为 {edge.weight}")
```

在这个示例中，通过不断选择权值最小且不形成环的边，最终得到连接所有网点所需的最少电线组合，即最小生成树。

1.6 贪心算法的局限性及替代方法

假设有以下一系列活动，每个活动都有开始时间和结束时间：

活动编号	开始时间	结束时间
活动 1	1	4
活动 2	3	5
活动 3	0	6
活动 4	5	7
活动 5	3	8
活动 6	5	9
活动 7	6	10
活动 8	8	11
活动 9	8	12
活动 10	2	14

问怎么安排这些活动能让它们尽量多地开展。

贪心算法的解决方案：

贪心算法通常会按照活动结束时间最早来选择。首先选择活动 1，因为它结束得最早。然后依次选择活动 4、活动 8 等。最终选择的活动为：活动 1、活动 4、活动 8 ，最多三个。

动态规划的解决方案：

```
# 假设活动的开始时间和结束时间分别存储在两个列表中，例如：
start_times = [1, 3, 0, 5, 3, 5, 6, 8, 8, 2]
end_times = [4, 5, 6, 7, 8, 9, 10, 11, 12, 14]

n = len(start_times)
dp = [0] * (n + 1)
dp[0] = 0

for i in range(1, n + 1):
if i == 1:
dp[1] = 1
elif i == 2:
if end_times[0] > start_times[1] or end_times[1] < start_times[0]:
dp[2] = 1
elif i == 3:
if (end_times[0] <= start_times[2] and end_times[2] >= start_times[0]) or \
(end_times[1] <= start_times[2] and end_times[2] >= start_times[1]):
dp[3] = 2
# 这里后续还需要继续完善代码以处理更多活动的情况，比如i > 3的情况，需要遍历前面的活动
# 判断每个活动与当前活动是否重叠，根据重叠情况更新dp[i]的值

print(dp)
```

最终得到 $\mathrm{dp}[10] = 4$ ，选择的活动为：活动 1、活动 4、活动 7、活动 9 ，最多四个，

通过比较可以发现，贪心算法选择了 3 个活动，而动态规划选择了 4 个活动，动态规划得到的结果更优，能够选择到更多相互兼容的活动，实现了全局最优。

通过以上的例子可以看出，贪心算法虽然在某些情况下能够高效地解决问题，但也存在明显的局限性。其最大的问题在于，通过一系列局部最优选择并不一定能保证获得全局最优解。

当贪心算法无法适用时，我们可以考虑以下几种思想方法：

动态规划：将复杂问题分解为多个重叠的子问题，并保存子问题的解以避免重复计算。通过逐步求解子问题，最终得出原问题的最优解。动态规划适用于具有最优子结构和重叠子问题的情况。

回溯算法：系统地尝试问题的所有可能解。当某一选择导致不满足条件或不是最优解时，回溯并尝试其他选择。回溯算法常用于解决组合优化等问题。

分支限界法：类似于回溯法，但通过评估和剪枝减少不必要的搜索，提高搜索效率。它常用于求解优化问题的最优解。

模拟退火算法：基于概率的随机搜索算法，模拟固体退火过程，在搜索过程中以一定概率接受次优解，有助于跳出局部最优解，趋向全局最优解。

遗传算法：模拟生物进化过程，通过遗传、变异和选择操作来搜索最优解。适用于复杂的优化问题，尤其是那些难以用传统方法精确求解的问题。

这些方法各有特点和适用场景，在面对具体问题时，需要根据问题的性质和要求选择合适的方法来获取全局最优解。

1.7 总结

贪心算法是一种具有特定适用场景的算法策略，它在一些问题中能够快速给出近似最优解，但也因其局限性并非适用于所有情况。克鲁斯卡尔算法展示了贪心思想在特定问题中的有效应用。当贪心算法失效时，我们有多种替代方法可供选择，如动态规划、回溯算法、分支限界法、模拟退火算法和遗传算法等。了解这些算法的特点和适用范围，能够帮助我们在面对不同问题时做出更合适的选择，以更有效地求解复杂的优化问题，获得真正的全局最优解。