哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

哈尔滨工业大学课程报告

为大规模知识图谱维护动态索引：一种基于负载预测的系统 KGWB

课 程 名 称： 大数据计算基础

院 (系) 名 称： 计算学部

专 业 班 级： 数据科学与大数据技术

学 生 姓 名： 王翰坤

学 生 学 号： 1183710106

二〇二一年一月

摘　　要

大规模知识图谱的查询效率与知识图谱上建立的索引密切相关。随着知识图谱数据和查询工作负载的不断变化，最初建立的索引可能不能很好地满足更新后的知识图谱数据及其负载。因此，需要感知动态变化的数据及负载，调整其索引。

本文提出了一种基于负载预测的知识图谱索引自动动态维护的系统 KGWB。该系统它针对大规模知识图谱系统的存储和查询特点，基于历史到达率时间模式，先对历史负载中的可索引数据库表列进行聚类。接着，系统对每个列簇构建 LSTM 时间序列模型来预测各个簇的未来到达率，并基于这个预测进行动态的索引调整，兼顾了系统查询性能以及负载预测和索引维护的成本。

关键词：知识图谱；数据库索引；负载预测；循环神经网络；无监督聚类

目　　录

第 1 章　绪论

1.1　研究问题的背景

由于大数据的兴起和计算能力的升级，业界涌现出以知识图谱（Knowledge Graph）为代表的一批大数据时代的产物。另一方面，大数据时代，数据规模庞大、数据管理应用场景复杂，传统数据库和数据管理技术面临很大的挑战。人工智能技术因其强大的学习、推理、规划能力，为包括知识图谱在内的各类数据库系统提供了新的发展机遇。人工智能赋能的数据库系统通过对数据分布、查询负载、性能表现等特征进行建模和学习，自动地进行查询负载预测、数据库配置参数调优、数据分区、索引维护、查询优化、查询调度等，以不断提高数据库针对特定硬件、数据和负载的性能。

大规模知识图谱的查询一直受到学术界和工业界的广泛关注，而其查询效率与知识图谱上建立的索引密切相关。而随着知识图谱数据的不断更新，以及其上查询工作负载的不断变化，最初建立的索引可能不能很好地满足更新后的知识图谱数据及其负载。因此，需要根据动态变化的数据及负载调整知识图谱索引。

本文就主要关注在大规模知识图谱上建立索引和动态维护索引的问题。

1.2　研究问题的挑战

本文面临的主要挑战有：

- 如何选取适当的性能评估函数，使得系统能够判断知识图谱及其负载对知识图谱查询性能的影响；
- 如何感知不断变化但又存在内在规律的的知识图谱负载等信息；
- 如何利用动态感知的结果，调整和维护知识图谱索引；
- 知识图谱的属性表存储中，每个表都有大量的列（例如，仅 DBpedia 的 album 一个表中，就有多达 102 个列），对每个列单独建模和维护索引是非常低效甚至不现实的；
- 索引维护算法要能够提升知识图谱的查询性能，同时保证调整索引的代价不宜过大。

1.3 当前研究工作的不足

目前，知识图谱的负载预测和是索引维护方面的研究还较为初步。

先前的工作研究了不同情境中的数据库工作负载建模。例如，一种方法是为系统的资源需求建模，而非工作负载本身的直接表示 [1, 2]。另一些方法通过回答关于 OLTP 工作负载变化的“What-if”问题来模拟 DBMS 的性能 [3, 4]，从而将工作负载建模为具有固定比率的不同类型事务的混合。早期的工作还使用预定义事务类型和到达率的数据库工作负载进行了建模 [5]。

但是，上述这些方法都有不足之处，使其不足以实现自动的索引调整。例如，某些方法使用的是有损压缩方案，该方案仅维护高级别的统计信息（如平均查询延迟和资源利用率）[1–4]。还有一些方法则假定工作负载是静态的，这样就无法捕获查询量和工作负载趋势随时间的变化 [5]。

此外，所有上述方法都是基于一般 SQL 数据库的，无法直接用于知识图谱，没有基于知识图谱的存储管理做有针对性的处理和优化。

1.4 本文提出的方法

本文基于提出了一种基于负载预测的知识图谱索引自动动态维护的方法KGWB。本方法基于历史到达率时间模式，不断地对用于存储在关系数据库中的可索引数据库表列进行聚类。接着，系统将构建 LSTM 模型来预测各个簇的未来到达率，并基于这个预测进行动态的索引调整。

1.5 本文的贡献

与先前的方法相比，本文的方法的主要优势在于，它针对大规模知识图谱系统的存储和查询特点，实现了不同类型负载的自动聚合，在预测准确度没有明显下降的情况下，大大减少了建模和训练的时间。此外，基于负载预测进行索引调整，不仅兼顾了系统查询性能以及预测和调整的成本，而且不受平台和硬件的限制。

1.6 章节安排

本文其余各章安排如下。第 2章是本文方法的框架性描述，第 3章、第 4章和第 5章分别是大规模知识图谱技术、无监督聚类技术和循环神经网络技术的介绍，第 6章详细介绍了实验的主要流程和实验结果，第 7章是相关工作的简介，第 8章是结论。

第 2 章　方法框架

2.1　概述

2.1.1　方法概要

本文提出一种基于负载预测的知识图谱动态索引维护系统 KGWB（Knowledge Graph Workload Bot），其查询效率相比静态索引有显著提升。

本文假定知识图谱的存储方案为属性表（property table）（见第 3 章），且 SPASQL 查询都已转化为等价的 SQL 查询。

系统由 4 个功能依次相接的主要模块组成：

- **预处理模块**。读取历史负载，从中提取和统计需要索引的表和列。
- **聚类模块**。将历史到达率类似的列聚合到一起，形成若干列簇。
- **预测模块**。对各列簇单独用循环神经网络建模，预测未来一段时间的到达率。
- **执行模块**。基于预测到达率执行实际负载，并动态维护和调整数据库的索引。

系统示意图如图2.1所示。实际应用中，预处理模块、聚类模块和预测模块每隔一段时间运行一次（可独立于执行数据库查询的服务器并行运行），获取未来一段时间的负载预测。基于这个预测，执行数据库查询的服务器动态地维护和调整数据库的索引，从而达到提升查询效率的目的。

2.1.2　动机与合理性分析

本方法的合理性基于以下几个观察或事实。

第一，在实际应用中，数据库的负载总体上是有规律和一定的可预测性的。例如，图2.2[6] 就展示了三种常见的负载规律模式。(a) 的 BusTracker 应用（一种公交路线查询与跟踪应用）的查询负载展现出的是以天为周期的周期性模式，这种模式也是最为常见和普遍的一种负载模式；(b) 的 Admissions 应用（一种面向学校师生的作业提交应用）表现出前期负载不断增长，在某一个时间点（如 deadline）达到顶峰后迅速下降的模式；(c) 的 Mooc 应用（一种线上教学平台）的负载则出现了一些突变（可能在新功能发布等情形下出现）。因此，我们可以根据历史负载对未来负载进行预测。

第二，尽管具体查询千变万化，涉及到的数据库表列（对应于知识图谱中的关系和属性）总量巨大，但它们并不是随机、独立地出现的。许多查询和所需索引的

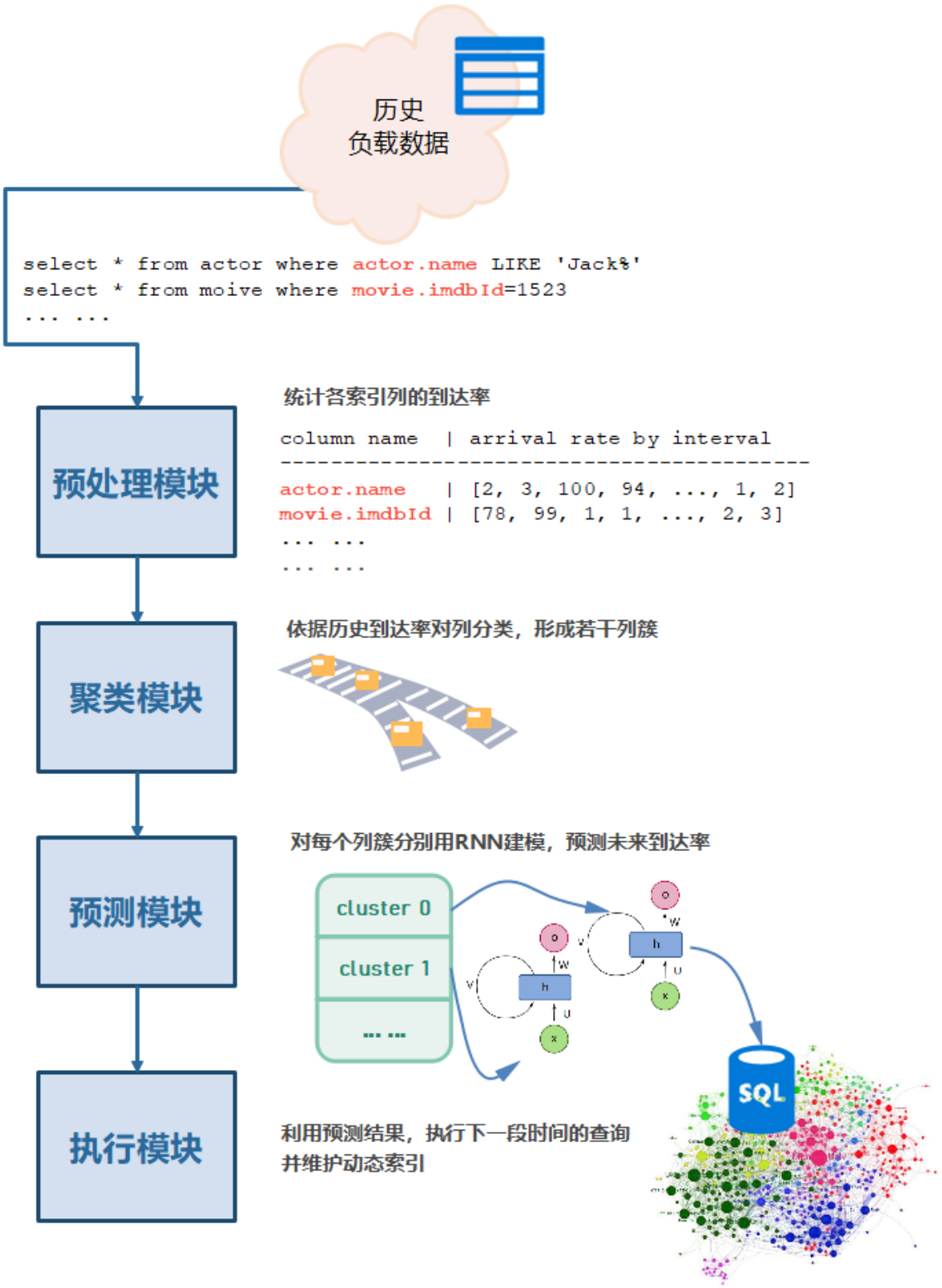


图 2.1　KGWB 系统示意图

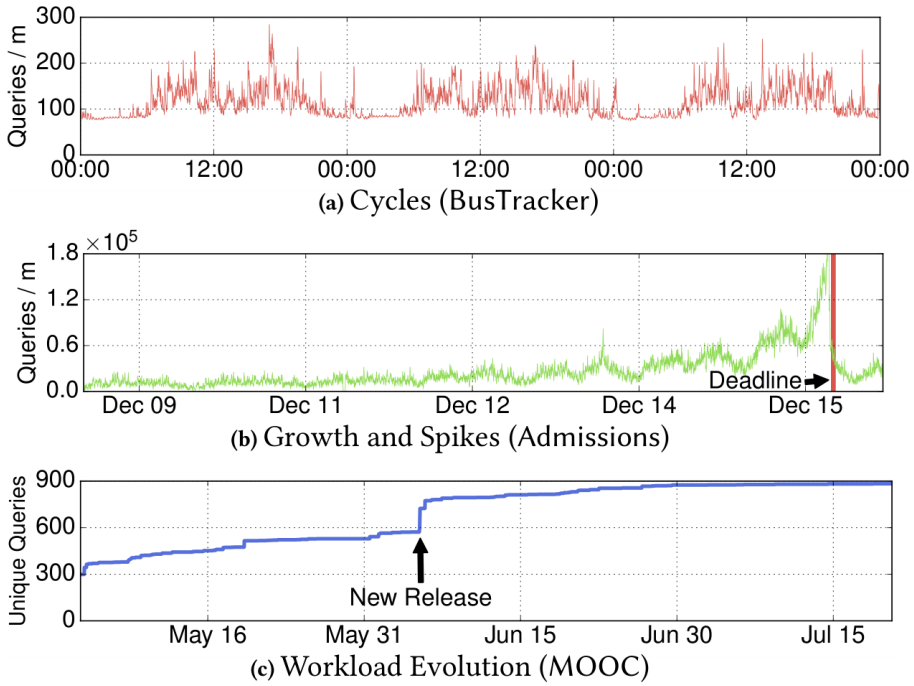


图 2.2 常见的负载规律模式

表列的出现具有很强的关联性，因此可以把到达率相近的那些列用无监督聚类算法聚合到一起，每个类训练一个模型用于未来负载的预测。

第三，数据库的查询性能很大程度上依赖于其索引的选择。对于 SELECT 查询来说，相比于无索引的情形，如果需要查询的列上已经建立了索引，系统的查询处理速度就能得到极大的提高。而对于 INSERT 和 DELETE 等需要改动数据表的查询来说，如果表上有列建立了索引，就要改动数据的同时更新索引，造成处理速度下降。另一方面，在数据量很大时，索引占用的空间也是不容忽视的。因此，有必要采用一些启发式的方法对数据库上的索引进行动态的维护和更新，使得所建立的索引既能够提升数据库查询性能，又不至于占用过多系统资源，有效维持二者之间的平衡。

2.2 各模块介绍

2.2.1 预处理模块

预处理模块读取已经转化为 SQL 语句的知识图谱查询（细节在 3中叙述），并提取出其中需要索引的数据库表列，连同查询时间戳等信息重新打包输出，如算法2.1所示。

其中，第 4 行最后的 *Explain* 的功能是解析给定查询可索引的列，多数主流数据库系统都对此提供了支持。例如，图3.7所示的查询中，WHERE 子句所提到的

算法 2.1: 预处理模块

Procedure *Preprocessing(workload)*

输入: 历史负载（一系列查询时间戳和查询字符串的二元组）

输出: 时间戳、查询串和需索引列的三元组的列表

1 $tripes \leftarrow \emptyset$

2 **for** $(timestamp, query) \in workload$ **do**

3 $tripes$.append($timestamp, query, Explain(query)$)

4 **return** *tripesList*

director.birthDate、director.networth、director.directs、actor.acts_in 和 movie.subject 都是可索引的列。

2.2.2 聚类模块

聚类模块读取预处理模块的结果，并按历史到达率对列集合做聚类。主要步骤如下（为方便叙述，设总的数据表列数为 n，历史负载时间跨度为 t）。

第一步，读取预处理模块的结果，并按时间段统计每个列的出现次数（或说到达率（arrival rate））：若时间粒度长度为 v（典型取值如一天），则本步将形成一张 n 行 t/v 列的二维统计表，位置 (i, j) 上的数值表示第 i 个数据库表列在第 j 个时间段上的出现次数。

第二步，选取不同的聚类数目 K，运用 K-Means 算法尝试对 n 个数据库表列进行聚类。这里，K 的选择范围是根据历史负载的特点人为选定的；而 K 的具体取值是通过枚举和多轮重复运行，用轮廓系数（silhouette score）（也可以利用肘部法则）自动选取的。

第三步，选择聚类效果最好的 K 的取值，对 n 个数据库表列进行最终聚类，并将聚类结果连同历史到达率统计表一起作为本模块的输出。

本模块流程如算法2.2所示。其中，第 16 行的判断条件可以改写成肘部法则对应的判断条件。

2.2.3 预测模块

预测模块读取各列簇的历史到达率，并对每个簇单独建立长短期记忆网络（LSTM）模型，用以预测未来的到达率。主要步骤如下。

第一步，合并列簇信息，对每个列簇，计算其中列的每天到达率的平均值，作为该列簇的历史到达率。

第二步，为模型训练做数据切分。对每个列簇，将每段连续一段时间粒度 W_h

算法 2.2: 聚类模块

Procedure *Clustering(tripes*(*timestamp*, *query*, *columns*)*)*

输入: 时间戳、查询串和需索引列的三元组的列表
输出: 数据库表列的聚类和历史到达率统计结果

Initalize v
$t \leftarrow \max(timestamp) - \min(timestamp)$
$T \leftarrow$ a table of size $n \times t/v$, filled by 0's
for $i \leftarrow 1$ **to** t/v **do**
    for $c \leftarrow 1$ **to** n **do**
        Count the occurrences o of column c in the i-th time interval
        $T[c][i] \leftarrow o$
Initalize r, m
$S \leftarrow -1$
for $k \leftarrow r.left$ **to** $r.right$ **do**
    $s \leftarrow -1$
    for $i \leftarrow 1$ **to** m **do**
        Execute K-Means algorithm by a clustering paramter K and randomly choosed inital center points
        $s \leftarrow \max(s,$ silhouette score of this clustering$)$
    if $S < s$ **then**
        $S \leftarrow s$
        $K \leftarrow k$
Do K-Means clustering by the finest clustering paramter K
$C \leftarrow$ the clustering results
return *C, T*

（如一周 7 天）的历史到达率作为输入，其后一个时间粒度的负载作为标签，切分出 Ot/v 组训练数据。

第三步，对每个列簇，建立一个 n-gram LSTM 时间序列预测模型并训练之，用以依据其过去一段时间的到达率预测未来到达率。

第四步，利用训练得到的模型预测未来一段时间各个列簇的到达率。设预测窗口的时间粒度数目为 W_f，由于模型模型每次只能预测接下来一个时间粒度的到达率，所以要所以要把预测值作为真实数据值输入模型，并迭代运行模型 W_f 次。

第五步，将列簇的预测结果缩放回各个列。

本模块流程如算法2.3所示。

2.2.4 执行模块

本模块利用每列的预测到达率，执行知识图谱负载查询并动态调整索引。为了有效平衡各类查询的性能和存储成本，本文采用一种启发式的方法 [7]，并设定

算法 2.3: 预测模块

Procedure *Prediction(C, T)*

输入: 聚类结果和到达率统计表

输出: 各个需要索引的数据库表列的未来到达率预测

1 $K \leftarrow$ the number of clusters
2 $u \leftarrow t/v$
3 Initalize W_h, W_f
4 **for** $c \leftarrow 1$ **to** K **do**
5 $Data_c \leftarrow$ an empty list
6 $T_c \leftarrow$ average arrival rate of all columns in cluster c
7 **for** $i \leftarrow 1$ **to** $u - W_h$ **do**
8 $Data_c$.append($(T_c[i \,..\, i + W_h - 1], T_c[i + W_h])$)
9 $M \leftarrow$ a initialized LSTM model
10 Train M with $Data_c$
11 **for** $i \leftarrow 1$ **to** W_f **do**
12 $p \leftarrow$ predict $T_c[u + i]$ with $T_c[u + i - w \,..\, u + i - 1]$
13 $T_c[u + i] \leftarrow p$
14 **for** $col \in cluster\ c$ **do**
15 $T[col][u + 1 \,..\, u + W_f] \leftarrow$ scaled $T_c[u + 1 \,..\, u + W_f]$
16 **return** $T[1 \,..\, n][u + 1 \,..\, u + W_f]$

了索引数目的上限。主要步骤如下。

第一步，丢弃数据表中原有的所有非主键索引，读取各列的预测到达率。

第二步，依次执行 W_f 个时间粒度内的所有查询。每处理完一个时间粒度，就随机丢弃一些索引（因为有最大索引数量限制），并在预测负载中下一个时间粒度出现得最频繁的若干列上新建索引。

第三步，记录实际的负载、执行时间和索引，为下一周期的预测做准备。

本模块流程如算法2.4所示。

算法 2.4: 执行模块

Procedure *Execute(T_p)*

输入: 各个需要索引的数据库表列的未来到达率预测

输出: /

1 Drop all old indexes on the database
2 **for** $i \leftarrow 1$ **to** W_f **do**
3 Build indexes on the most frequent columns in the prediction on day i, that is (T_p[?][i])
4 Execute the actual queries of day i
5 Record the actual workload, processing time and indexes it used
6 Randomly drop some indexes

第 3 章　大规模知识图谱

3.1　概述

知识图谱是人工智能的重要基石，各领域大规模知识图谱的构建和发布对知识图谱数据管理提出了新的挑战。

知识图谱的概念由谷歌 2012 年正式提出，旨在实现更智能的搜索引擎，并且于 2013 年以后开始在学术界和业界普及。目前，随着智能信息服务应用的不断发展，知识图谱已被广泛应用于智能搜索、智能问答、个性化推荐、情报分析、反欺诈等领域。另外，通过知识图谱能够将 Web 上的信息、数据以及链接关系聚集为知识，使信息资源更易于计算、理解以及评价，并且形成一套 Web 语义知识库。知识图谱以其强大的语义处理能力与开放互联能力，可为万维网上的知识互联奠定扎实的基础，使 Web 3.0 提出的“知识之网”愿景成为了可能。

知识图谱作为符号主义发展的最新成果，是人工智能的重要基石。随着知识图谱规模的日益扩大，其数据管理问题愈加重要。一方面，以文件形式保存知识图谱无法满足用户的查询、检索、推理、分析及各种应用需求; 另一方面，传统关系数据库的关系模型与知识图谱的图模型之间存在显著差异，关系数据库无法有效管理大规模知识图谱数据。为了更好地管理知识图谱，语义 Web 领域发展出专门存储 RDF 数据的三元组库；数据库领域发展出用于管理属性图的图数据库。但是目前还没有一种数据库系统被公认为是具有主导地位的知识图谱数据库。

目前，规模为百万顶点和上亿条边的知识图谱数据集已经常见。例如，本文实验中所使用的维基百科知识图谱 DBpedia 就有超过 30 亿条数据；其他典型的大规模知识图谱还有地理信息知识图谱 LinkedGeoData（>30 亿条）和蛋白质知识图谱 UniProt（>130 亿条）等。各领域大规模知识图谱的构建和发布对知识图谱数据管理提出了新的挑战。

3.2　RDF 图数据模型

数据模型定义了数据的逻辑组织结构、其上的操作和约束，决定了数据管理所采取的有效方法与策略，对于存储管理、查询处理、查询语言设计均至关重要。可以认为，知识图谱数据模型是图数据模型的继承和发展。知识图谱数据模型基于图结构，用顶点表示实体，用边表示实体间的联系，这种一般和通用的数据表示恰好能够自然地刻画现实世界中事物的广泛联系。目前，主流的知识图谱数据模

型有 RDF 图模型和属性图模型。本文仅介绍可以基于传统关系 SQL 数据库管理系统的 RDF 图模型。

RDF 全称为资源描述框架（Resource Description Framework），是万维网联盟制定的在语义 Web（semantic Web）上表示和交换机器可理解（machine-understandable）信息的标准数据模型。

在 RDF 图中，每个资源具有一个 HTTP URI 作为其唯一 ID；RDF 图定义为三元组 (s, p, o) 的有限集合；每个三元组表示是一个事实陈述句，其中 s 是主语，p 是谓语，o 是宾语；(s, p, o) 表示 s 与 o 之间具有联系 p，或表示 s 具有属性 p 且其取值为 o。

图3.1所示的 RDF 图示例描述了一个电影知识图谱，其中包括电影（movie）Titanic、导演（director）James_Cameron、演员（actor）Leonardo_DiCaprio 和 Kate_Winslet 等资源，这些资源上的若干属性，以及资源之间的执导（direct）和出演（acts_in）联系。

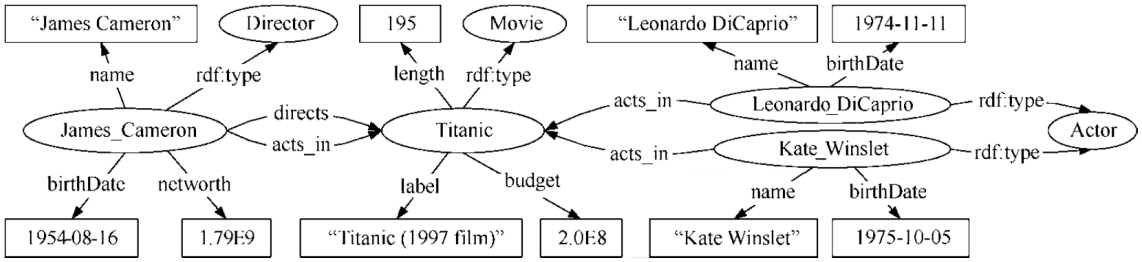


图 3.1 RDF 图示例：电影知识图谱

在 RDF 图示例中，椭圆表示资源主体（Subject），用矩形表示字面量（Literal）；一条有向边及其连接的两个顶点对应于一条三元组，尾顶点是主语，边标签是谓语，头顶点是宾语。资源所属的类型由 RDF 内置谓语 rdf:type 指定，如三元组 (James_Cameron, rdf:type, Director) 表示 James_Cameron 是导演. 本例中还有几种类型的字面常量：字符串放在双引号中；整数 195 是电影分钟数；浮点数 1.79E9 和 2.0E8 分别是导演净资产和电影预算金额；日期 1954-08-16、1974-11-11 和 1975-10-05 分别是导演和两位演员的出生日期。

从数据模型角度看，RDF 图是一种特殊的有向标签图（Directed Labeled Graphs）。W3C 还为 RDF 图定义了基于描述逻辑（一阶谓词逻辑的可判定子集）的本体语言 OWL，可描述概念之间更为复杂的逻辑关系，并给出了相应的逻辑推理规则。

3.3 知识图谱的查询语言

知识图谱查询语言主要实现了知识图谱数据模型的操作部分。目前，RDF 图的标准查询语言是 SPARQL。

SPARQL 是 W3C 制定的 RDF 知识图谱标准查询语言。SPARQL 从语法上借鉴了 SQL。SPARQL 查询的基本单元是三元组模式（triple pattern），多个三元组模式可构成基本图模式（basic graph pattern）。SPARQL 支持多种运算符，将基本图模式扩展为复杂图模式（complex graph pattern）。

例如，在图3.1所示的电影知识图谱上，可以做如图3.2所示的查询，它的含义是“查询 1950 年之后出生的资产大于 1.0E9 的导演执导的电影的出演演员”。

```
SELECT ?x4 ?x5
WHERE {?x1 rdf:type Director.?x1 birthDate?x2.FILTER (?x2>=1950-01-01).
       ?x1 networth?x3.FILTER(?x3>1.0E9).?x1 directs?x4.?x5 acts_in?x4.}
```

图 3.2 SPARSQL 示例查询

查询结果如表3.1所示。

?x4	?x5
Titanic	Leonardo_DiCaprio
Titanic	Kate_Winslet
Titanic	James_Cameron

表 3.1 SPARSQL 查询结果

SPARQL 经过 W3C 的标准化过程，具有精确定义的语法和语义。这个例子中，SELECT 子句指明要返回的结果变量；WHERE 子句指明查询条件；?x1 rdf:type Director. 是三元组模式，其中，rdf:type 和 Director 是常量，?x1 是变量（SPARQL 中的变量以? 开头)，句点表示一条三元组模式的结束。

3.4 知识图谱存储管理

知识图谱的存储管理大体上也分为两种思路：一是基于传统关系数据库的方法，二是原生知识图谱数据库专门为知识图谱而设计的底层存储管理方案。关系数据库目前仍是使用最多的数据库管理系统。基于关系数据库的存储方案是目前知识图谱数据的一种主要存储方法。按照时间发展顺序，依次出现了以下的基于关系的知识图谱存储方案，包括：三元组表、水平表、属性表、垂直划分、六重索引和 DB2RDF 等。本节主要介绍三元组表、水平表和本文实验中所使用的属性表方案。

3.4.1 三元组表

三元组表（triple table）是将知识图谱存储到关系数据库的最简单、最直接的办法，就是在关系数据库中建立一张具有 3 列的表，模式为

triple_table(subject, predicate, object).

这 3 列分别表示主语、谓语和宾语；将知识图谱中的每条三元组存储为三元组表 triple_table 中的一行记录。RDF 数据库系统 3store 使用的就是三元组表存储方案。

表3.2展示了图3.1所示电影知识图谱对应的三元组表的前 7 行。

subject	predicate	object
James_Cameron	type	Director
James_Cameron	name	J̆ames Cameron
¨James_Cameron	birthDate	1954-08-16
James_Cameron	networth	1.79E9
James_Cameron	directs	Titanic
James_Cameron	acts_in	Titanic
Titanic	type	Movie
…	…	…

表 3.2 三元组表示例

三元组表存储方案虽然简单明了，但其行数过于庞大（与知识图谱的边数相等），这导致的最大问题在于将知识图谱查询翻译为 SQL 查询后会产生三元组表的大量自连接操作。

例如，3.2的 SPARQL 查询翻译为等价的 SQL 查询后如3.3所示。一般，自连接（self-join）的数量与 SPARQL 中三元组模式数量相当。当三元组表规模较大时，多个自连接操作将影响 SQL 查询性能。此外，由于三元组表方案形成的数据库将所有图谱信息装载在一个表 3 个列中，所以在这张表上建立索引将耗费极大的时空资源，且优化余地很小。

```
SELECT t4.object, t5.subject
FROM t AS t1, t AS t2, t AS t3, t AS t4, t AS t5
WHERE t1.subject=t2.subject AND t2.subject=t3.subject AND t3.subject=t4.subject
  AND t4.object=t5.object AND t1.predicate='rdf:type' AND t1.object='Director'
  AND t2.predicate='birthDate' AND t2.object>='1950-01-01' AND t3.predicate='networth'
  AND t3.object>1.0E9 AND t4.predicate='directs' AND t5.predicate='acts_in'
```

图 3.3 基于三元组表存储的 SQL 查询

3.4.2 水平表

水平表（horizontal table）存储方案同样非常简单。水平表的每行记录存储知识图谱中一个主语的所有谓语和宾语。实际上，水平表相当于知识图谱的邻接表。

水平表的列数是知识图谱中不同谓语的数量，行数是知识图谱中不同主语的数量。早期的数据库系统 DLDB 就是使用的这种存储方式。图3.4展示了图3.1所示电影知识图谱对应的水平表。

subject	type	name	birthDate	networth	label	budget	length	directs	acts_in
James_Cameron	Director	James Cameron	1954-08-16	1.79E9				Titanic	Titanic
Titanic	Movie				Titanic (1997 film)	2.0E8	195		
Leonardo_DiCaprio	Actor	Leonardo DiCaprio	1974-11-11						Titanic
Kate_Winslet	Actor	Kate Winslet	1975-10-05						Titanic

图 3.4 水平表示例

在水平表存储方案中，图3.2中的 SPARQL 查询可以等价地翻译为3.3中的 SQL 查询。可见，与三元组表相比，水平表存储方案使得查询大为简化，自连接操作由 4 个减少到 2 个。

```
SELECT t2.subject,t3.subject
FROM t AS t1,t AS t2,t AS t3
WHERE t1.type='Director' AND t1.birthDate>='1950-01-01' AND t1.networth>1.0E9
  AND t1.directs=t2.subject AND t3.acts_in=t2.subject
```

图 3.5 基于水平表存储的 SQL 查询

水平表的缺点主要在于：（1）所需列的数目等于知识图谱中不同谓语数量，在真实知识图谱数据集中，不同谓语数量可能为几千个到上万个，很可能超出关系数据库所允许的表中列数目上限；（2）对于一行来说，仅在极少数列上具有值，表中存在大量空值，空值过多会影响表的存储、索引和查询性能；（3）知识图谱的更新往往会引起谓语的增加、修改或删除，即水平表中列的增加、修改或删除，这是对于表结构的改变，成本很高。

3.4.3 属性表

属性表（property table）存储方案是对水平表的细分，将同类主语存到一个表中，解决了表中列数目过多的问题。著名的知识图谱计算框架 Apache Jena 就是使用的属性表存储方案。

图3.6展示了图3.1所示电影知识图谱对应的属性表。可以看到，属性表方案根据主语类型的不同，将水平表拆分成了 3 个表。

在属性表存储方案中，图3.2中的 SPARQL 查询可以等价地翻译为3.7中的 SQL 查询。可以看到，该查询与水平表上的等价查询（图3.5）相比，t1 变为了 director，t2 变为了 movie，t3 变为了 actor，由自连接转变为多表连接，而且每个表对应于一个类型，省去了类型的判断，提高了查询的可读性。

director

subject	type	name	birthDate	networth
James_Cameron	Director	James Cameron	1954-08-16	1.79E9

movie

subject	type	label	budget	length
Titanic	Movie	Titanic (1997 film)	2.0E8	195

actor

subject	type	birthDate
Leonardo_DiCaprio	Actor	1974-11-11
Kate_Winslet	Actor	1975-10-05

图 3.6 属性表示例

```
SELECT movie.subject,actor.subject
FROM director,movie,actor
WHERE director.birthDate>='1950-01-01' AND director.networth>1.0E9
  AND director.directs=movie.subject AND actor.acts_in=movie.subject
```

图 3.7 基于属性表存储的 SQL 查询

第 4 章　无监督聚类算法

4.1　概述

如第 2章所述，可能出现的查询种类和需索引的列都是极多的，为每种查询、每个数据库表列都单独建立和训练预测模型是不可行的。另外，知识图谱应用事先完全不知道会有什么样的查询、涉及到哪些列，更不可能对每种查询、每个列标注其分类。因此，系统需要采用无监督的聚类算法，依据历史到达率对历史负载查询中涉及到的数据库表列进行分类。

本章主要介绍无监督聚类算法的背景，并着重叙述本系统所使用的 *K*-Means 算法。

4.2　无监督学习方法

无监督学习是机器学习技术的一类，其用于发现数据中的模式（patterns）。无监督算法的输入数据是无标签、未手工标注的，也就是说，对于无监督算法，其只需提供输入，而无需提供对应的标签，——无监督学习算法自己去挖掘数据中有意义的结构信息。

如图4.1，左图是监督学习，可以使用回归技术寻找特征之间的最佳拟合；右图是无监督学习，输入是特征分离的，预测是基于其归属的聚类进行的.

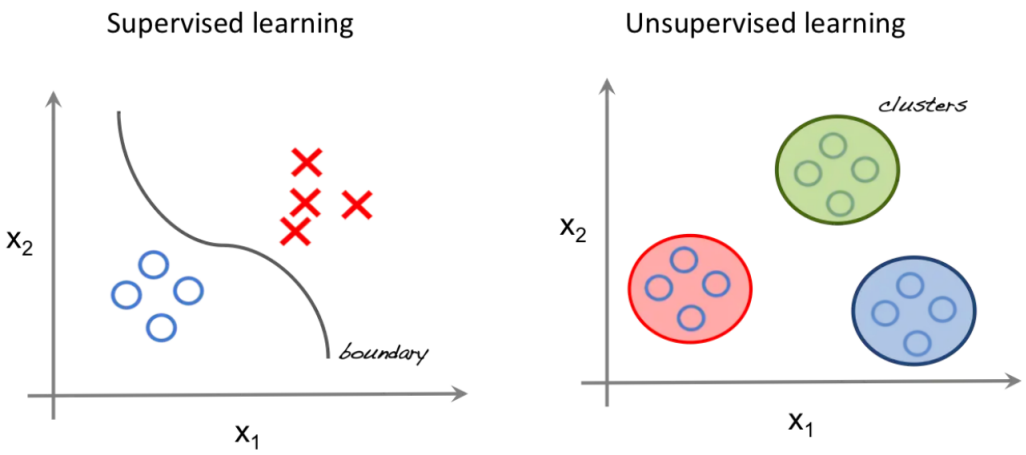


图 4.1　监督学习与无监督学习的对比

4.3 聚类问题与 *K*-Means 算法

4.3.1 聚类问题

聚类问题，即给定一个元素集合 *D*，使用某种算法将 *D* 划分成 *K* 个子集，要求每个子集内部的元素之间相异度尽可能低，而不同子集的元素相异度尽可能高。通常把这样得到的一个子集叫做一个簇。

与分类不同，分类是监督学习，要求分类前明确各个簇别，并断言每个元素映射到一个类别；而聚类是无监督学习，在聚类前可以不知道类别甚至不给定类别数量。目前聚类广泛应用于统计学和数据库技术等领域，主要算法有 *K*-Means、*K*-centroids 和 GMM-EMM 等。本系统所采用的是 *K*-Means（*K*-均值）算法。

4.3.2 *K*-Means 算法

4.3.2.1 算法流程

K-Means 算法就是根据某种度量方式（常用欧氏距离、余弦相似度等，度量距离越小，相关性越大），将相关性较大的一些样本点聚集在一起。*K*-Means 的过程为: 先在样本点中随意选取 *K* 个点作为暂时的聚类中心，然后依次计算每一个样本点与这 *K* 个点的距离，将每一个点与距离它最近的中心点聚在一起，形成 *K* 堆。接着, 求每个堆的均值，将求得的均值作为这个类的新的中心点。如此这般迭代下去，直至中心点不再改变停止。

事实上，*K*-Means 算法优化的损失函数是各个点到其类中心点的距离的最小值:

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\left\|\mathbf{x}_n - \boldsymbol{\mu}_k\right\|^2 .$$

对于每轮迭代，做如下两步。第一步，计算每个点所归属的中心点。这里，用 r_{nk} 这个示性变量表示点 *n* 是否属于第 *k* 类：

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \left\|\mathbf{x}_n - \boldsymbol{\mu}_j\right\|^2 \\ 0 & \text{otherwise.} \end{cases} .$$

确定了所有的 r_{nk} 之后，就把 *r* 矩阵视作固定的，尝试优化各类的中心点 $\boldsymbol{\mu}_k$。这也就是算法的第二步。这里，只要我们对每个 $\boldsymbol{\mu}_k$ 求偏导，就得到

$$2\sum_{n=1}^{N} r_{nk}\left(\mathbf{x}_n - \boldsymbol{\mu}_k\right) = 0,$$

从而有

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk}\mathbf{x}_n}{\sum_n r_{nk}}.$$

这就是 K-Means 算法的理论依据。

K-Means 算法主要流程如算法4.1所示。算法第 6 行中的 $1\left\{c^{(i)}=j\right\}$ 含义与 r_{ij} 相同。

算法 4.1: K-Means 算法

Procedure $KMeans(x^{(1)}, x^{(2)}, \ldots, x^{(n)})$

输入: n 个样本点

输出: 样本点集聚为 K 类

1 $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R} \leftarrow$ Randomly choose K cluster centroids

2 **while** *not convergence* **do**

3 **for** $i \leftarrow 1$ **to** n **do**

4 $c^{(i)} \leftarrow \arg\min_j \left\|x^{(i)}-\mu_j\right\|^2$

5 **for** $j \leftarrow 1$ **to** K **do**

6 $u_j \leftarrow \left(\sum_{i=1}^{m} 1\left\{c^{(i)}=j\right\} x^{(i)}\right) / \left(\sum_{i=1}^{m} 1\left\{c^{(i)}=j\right\}\right)$

7 **return** (c, μ)

4.3.3 参数 K 的选取

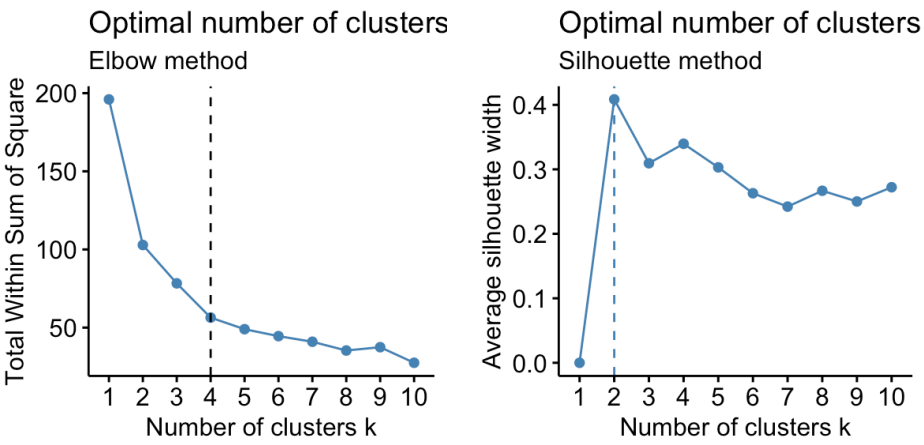
K-Means 等无监督分类算法的聚类数目 K 的选择是一个重要的话题。K 值无论太小或太大都会造成聚类效果不尽如人意，给后续处理造成麻烦。下面介绍两种选取 K 的方法。

4.3.3.1 肘部法则

常见的一种方法是肘部法则（Elbow Method）。

它要求先绘制一张聚类数量-WSS（Within cluster Sum of Squares，即各个点到簇中心的距离的平方的和）的折线图，然后将转折明显的聚类数量值设为最合适的 K。例如，如图4.2左图所示的折线图上，最佳的聚类 K 值就是 4。

手肘法的核心思想是：随着聚类数 K 的增大，样本划分会更加精细，每个簇的聚合程度会逐渐提高，那么误差平方和 WSS 自然会逐渐变小。并且，当 K 小于真实聚类数时，由于 K 的增大会大幅增加每个簇的聚合程度，故 WSS 的下降幅度会很大；而当 K 到达真实聚类数时，再增加 K 所得到的聚合程度回报会迅速变小，所以 WSS 的下降幅度会骤减，然后随着 K 值的继续增大而趋于平缓。这就表明，WSS 和 K 的关系图是一个接近于手肘的形状，而这个肘部对应的 K 值就是样本集的最佳聚类数。



(a) 肘部法则　　(b) 轮廓系数法

图 4.2　*K*-Means 算法对参数 *K* 的选择

4.3.3.2　轮廓系数法

该方法的核心指标是轮廓系数（Silhouette Coefficient），某个样本点 $x^{(i)}$ 的轮廓系数定义如下：

$$S = \frac{b_i - a_i}{\max(a_i, b_i)}$$

其中，a_i 是 $x^{(i)}$ 与同簇的其他样本的平均距离（即凝聚度）；b_i 是 $x^{(i)}$ 与最近簇中所有样本的平均距离（即分离度）。求出所有样本点的轮廓系数后再求平均值就得到了平均轮廓系数。显然，平均轮廓系数的取值范围为 $[-1, 1]$，且簇内样本的距离越近、簇间样本距离越远，平均轮廓系数越大，聚类效果也就越好。那么，很自然地，平均轮廓系数最大的 K 便是最佳聚类数。例如，如图4.2右图所示的折线图上，最佳的聚类 K 值就是 7。

第 5 章　循环神经网络

5.1　概述

本系统中需要依据历史负载预测数据库表列的未来到达率，这个预测工作交给循环神经网络模型完成。本章就主要介绍深度学习、人工神经网络和循环神经网络技术的相关内容。

5.2　深度学习

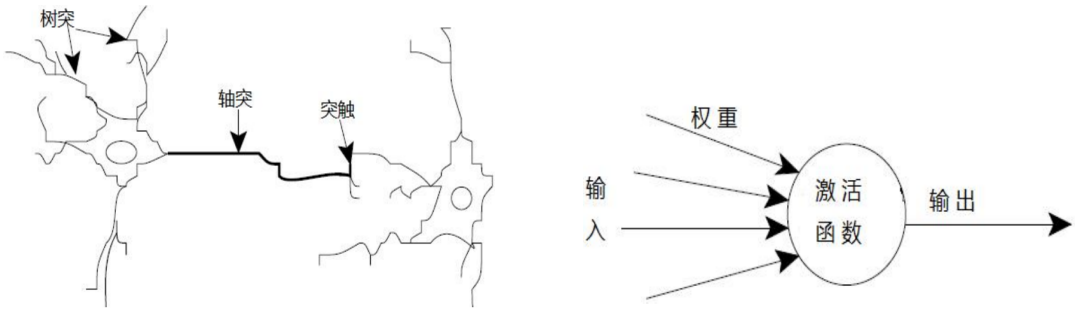
深度学习（Deep Learning, DL）机器学习的技术和研究领域之一，通过建立具有阶层结构的人工神经网络（Artificial Neural Networks, ANNs），在计算系统中实现人工智能。

深度学习所使用的阶层 ANN 具有多种形态，其阶层的复杂度被通称为“深度”。按构筑类型，深度学习的形式包括多层感知器、卷积神经网络、循环神经网络、深度置信网络等。深度学习使用数据对其构筑中的参数进行更新以达成训练目标，该过程被通称为“学习”。学习的常见方法为梯度下降算法及其变体，一些统计学习理论被用于学习过程的优化。在应用方面，深度学习被用于对复杂结构和大样本的高维数据进行学习，按研究领域包括计算机视觉、自然语言处理、生物信息学、自动控制等，且在人像识别、机器翻译、自动驾驶等现实问题中取得了成功。

5.3　人工神经网络

人工神经网络 ANN 是 20 世纪 80 年代以来人工智能领域兴起的研究热点。它从信息处理角度对人脑神经元网络进行抽象，建立某种简单模型，按不同的连接方式组成不同的网络。在工程与学术界也常直接简称为神经网络或类神经网络。

神经网络是一种运算模型，由大量的节点（或称神经元）之间相互联接构成。每个节点代表一种特定的输出函数，称为激励函数（activation function）。每两个节点间的连接都代表一个对于通过该连接信号的加权值，称之为权重，这相当于人工神经网络的记忆。网络的输出则依网络的连接方式，权重值和激励函数的不同而不同。而网络自身通常都是对自然界某种算法或者函数的逼近，也可能是对一种逻辑策略的表达。最近十多年来，人工神经网络的研究工作不断深入，已经取得了很大的进展，其在模式识别、智能机器人、自动控制、预测估计、生物、医



(a) 生物神经元　　(b) 人工神经元

图 5.1　人工神经元是生物神经元的抽象符号性概括

学、经济等领域已成功地解决了许多现代计算机难以解决的实际问题，表现出了良好的智能特性。

人工神经网络是一种旨在模拟人脑结构及其功能的信息处理系统，把自然的神经单元的复杂性进行了高度抽象的符号性概括，如图5.1所示。一个人工神经元包括多个输入（类似突出），通过与不同的权值相乘模拟信号输入的强度，然后用数学函数计算决定是否激发神经元，最后进行输出。人工神经网络即为大量人工神经元链接组成的信息处理模型，如图5.2所示。

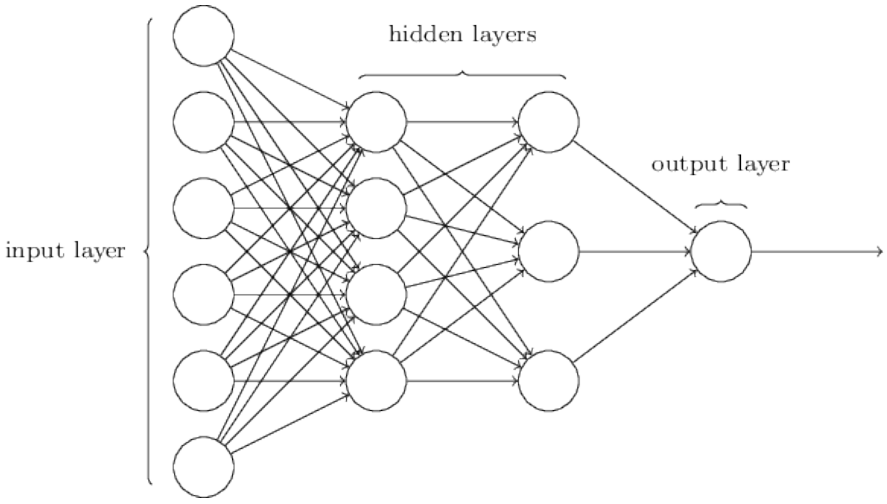


图 5.2　人工神经网络

5.4　循环神经网络

5.4.1　基础模型

循环神经网络（Recurrent Neural Network, RNN）是一类以序列（sequence）数据为输入，在序列的演进方向进行递归（recursion）且所有节点（循环单元）按链式连接的递归神经网络（recursive neural network）[8]。

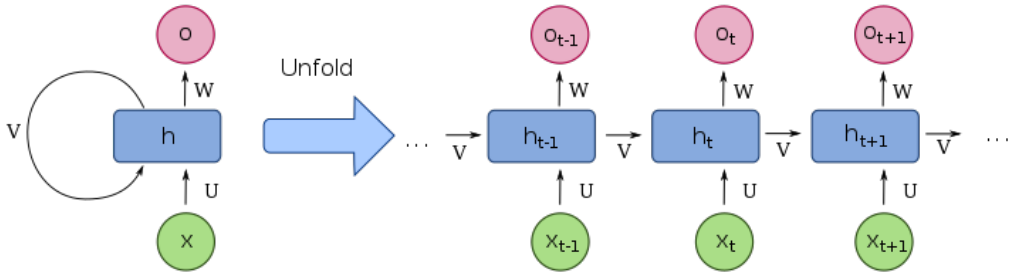


图 5.3 循环神经网络及其展开

对循环神经网络的研究始于二十世纪 80-90 年代，并在二十一世纪初发展为深度学习（Deep Learning）算法之一，其中双向循环神经网络（Bidirectional RNN, Bi-RNN）和长短期记忆网络（Long Short-Term Memory networks，LSTM）是常见的循环神经网络。循环神经网络具有记忆性、参数共享并且图灵完备（Turing completeness），因此在对序列的非线性特征进行学习时具有一定优势。循环神经网络在自然语言处理（Natural Language Processing, NLP），例如语音识别、语言建模、机器翻译等领域有应用，也被用于各类时间序列预报。因此，在本系统中，用 RNN 模型来做未来到达率的预测是很合适的。

如图5.3所示，RNN 同基础神经网络一样由输入层（Input Layer）、输出层（Output Layer）和隐藏层（Hidden Layer）构成。不过，基础的神经网络只在层与层之间建立了链接，而 RNN 在层之间的神经元之间也建立了权连接。循环体现在隐藏层中，隐藏层中的神经元之间也是带有权值的，也就是说随着序列的不断推进，前面的隐藏层将影响后面的隐藏层。

5.4.2 长短期记忆网络

5.4.2.1 基本结构

长短期记忆网络（Long Short-Term Memory, LSTM），是一种时间循环神经网络，适合于处理和预测时间序列中间隔和延迟相对较长的重要事件 [9]。

由于 LSTM 有很多的变种，这里我们以最常见的 LSTM 为例叙述。一个展开的 LSTM 的结构如图5.4所示。

LSTM 结构中，在每个序列索引位置 t 时刻向前传播的，除了和 RNN 一样的隐藏状态 h_t，还多了另一个隐藏状态，一般称为细胞状态（Cell State），记为 C_t。如图5.5所示。

LSTM 中还有门控结构（Gate）。LSTM 在每个序列索引位置 t 的门一般包括

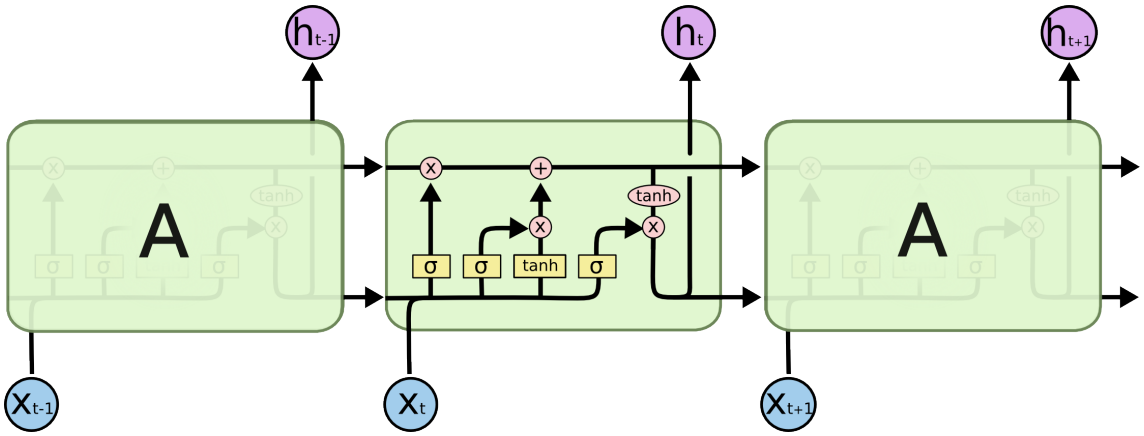


图 5.4　LSTM 单元的结构

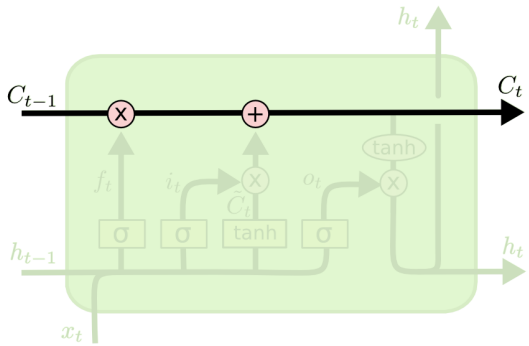


图 5.5　LSTM 单元的细胞状态

遗忘门、输入门和输出门三种。

5.4.2.2　遗忘门

遗忘门（forget gate）控制是否遗忘，即以一定的概率选择是否遗忘上一层的隐藏细胞状态。其子结构如图5.6所示。

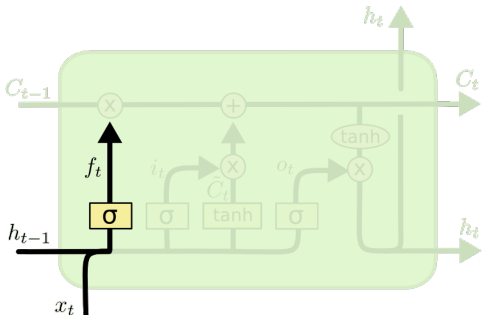


图 5.6　LSTM 单元的遗忘门

设图中上一层的隐藏状态为 h_{t-1}，本层输入为 x_t，W_t、U_t 和 b_f 是本单元遗忘门的系数和偏置，激活函数为 sigmoid 函数 σ，则 f_t 的表达式为

$$f_t = \sigma\left(W_f h_{t-1} + U_f x_t + b_f\right).$$

5.4.2.3 输入门

输入门（input gate）负责处理当前序列位置的输入，其子结构如图5.7所示。

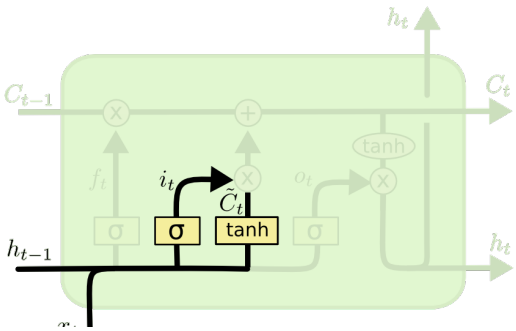


图 5.7 LSTM 单元的输入门

从图中可以看到输入门由两部分组成，第一部分使用了 sigmoid 激活函数，输出为 i_t , 第二部分使用了 tanh 激活函数，输出为 a_t。之后，两者的结果后面会相乘以更新细胞状态。输入门表达式为

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

其中 $W_i, U_i, b_i, W_a, U_a, b_a$ 是本单元输入门的系数和偏置。

5.4.2.4 细胞状态更新

前面的遗忘门和输入门的结果都会作用于细胞状态 C_t，图5.8展示了如何从 C_{t-1} 得到 C_t。

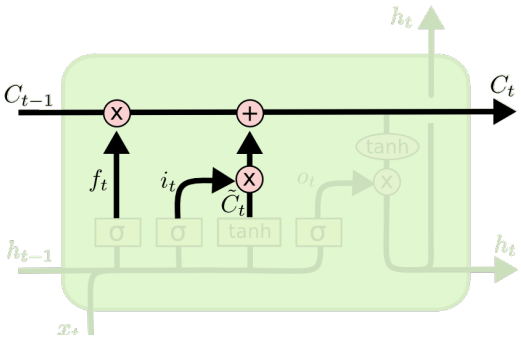


图 5.8 LSTM 单元的细胞状态更新

细胞状态 C_t 由两部分组成，第一部分是 C_{t-1} 和遗忘门输出与 $f(t)$ 的乘积，第二部分是输入门的 i_t 和 a_t 的乘积，即：

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t.$$

其中，$*$ 为 Hadamard 积，这是一种同维度向量之间的向量积，所得向量的每个元

素为被积向量在相同位置的元素乘积，即 $(a,b)*(c,d)=(ac,bd)$。

5.4.2.5 输出门

LSTM 单元的最后一个组件是输出门，其结构如图5.9所示。

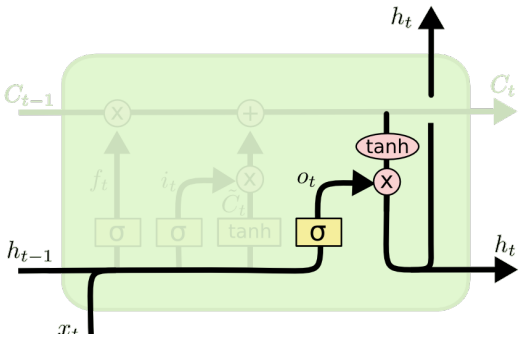


图 5.9 LSTM 单元的输出门

其表达式可以写为

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right).$$

其中 W_o, b_o 是本单元输出门的系数和偏置。

就这样，每个 LSTM 单元通过遗忘门，输入门，输出门三个部件舍弃或保留流过的信息，完成细胞状态的更新。

第 6 章　实验

6.1　概述

本章主要叙述实验的设计、过程和结果，并对实验结果和参数之间的关联做了分析。

6.2　实验环境

6.2.1　硬件环境

- HASEE GOD OF WAR Z7-KP7GZ
- Intel Core i7-8750H
- NVIDIA GTX 1060 (6G)
- 16G RAM

6.2.2　软件环境

- Windows 10 Professional (x64) 10.0.19041.746
- Python 3.7.9
- PyTorch 1.7.1 (with CUDA 10.2)
- MySQL 8.0.22
- PyCharm 2020.3 (Professional Edition)

6.3　实验设计

6.3.1　数据准备

6.3.1.1　DBpedia 简介

本文中采用 DBpedia[10] 作为原始数据。DBpedia 是一个从维基百科（Wikipedia）里萃取结构化内容的项目。这些项目所得的结构化信息，也将放在互联网中公开让人取阅。DBpedia 广纳了人类知识不同领域，十分多元的范畴资料，且允许用户查询跟维基百科相关资源之间的关系与性质。这使得它自然而然成为链接众多资料集的枢纽，让外部资料集能够链接到相关的概念。Bpedia 的资料集是跟其他许多网络上不同的开放资料资料集在 RDF 的层级交互相连着。透过这些资料集，可以让应用程序丰富 DBpedia 的资料。

DBpedia 也是一个很特殊的语义网应用范例，是世界上最大的多领域知识本体之一。它强化了维基百科的搜寻功能，并将其他资料集连结至维基百科。透过这样的语义化技术的介入，让维基百科的庞杂资讯有了许多创新而有趣的应用，例如手机版本、地图整合、多面向搜寻、关系查询、文件分类与标注等等。

6.3.1.2 数据载入与负载生成

本文中采用属性表方案存储 DBpedia 的海量数据。这是主要是因为属性表不仅空间利用率较高，而且实现较为简单。此外，其对应的 SQL 查询语句可读性强，易于解析，可以放便地提取处需索引的列。

实验使用 MySQL 存储 DBpedia 的属性表，并在其上运行工作负载。

实验的数据集为 DBpedia 的 infobox 数据集。原始数据集为三元组列表（n-triples）形式，需要先对其进行语义解析，并将相同 infobox（即3.4.3小节中提到的 rdf:type）的主体存放在一张数据表中。最终得到了 1678 个数据表，其中行数最多的 10 个表的信息如表6.1所示

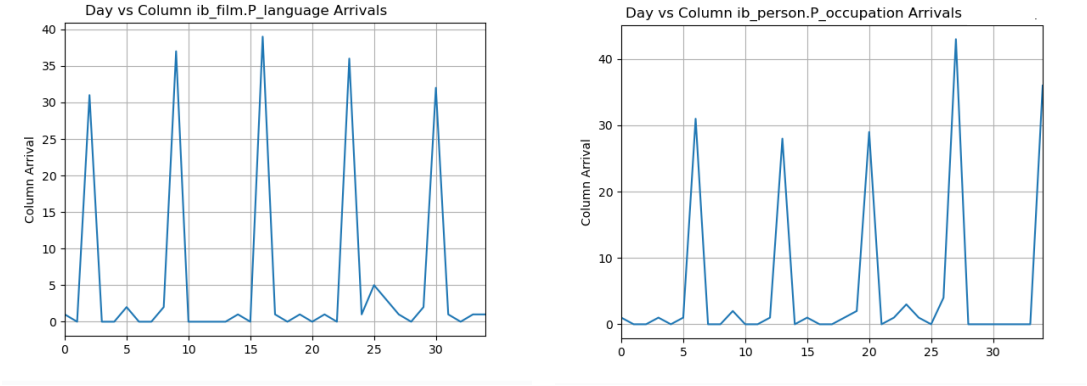
Infobox	Rows	Columns	Data Length (MB)
ib_album	51876	102	28.55
ib_film	20718	133	12.52
ib_musical_artist	12728	109	6.52
ib_single	12442	78	7.52
ib_book	9347	106	5.52
ib_cvg	8778	59	5.52
ib_company	8737	156	4.52
ib_actor	8305	95	3.52
ib_television_episode	6764	111	4.52
ib_cityit	6236	78	3.52

表 6.1 属性表存储中行数最多的 10 个表

负载是通过模拟周期性随机生成的。由于机器算力有限，我设为每 5 分钟出现一条查询，共生成了 5 周的负载数据，其中总计 10080 条查询。其中前 4 周的查询当作“历史负载”，用于预测第 5 周负载的预测。我将所有可索引列划分成 7 个不相交的子集，分别用于周一、周二至周日。每天查询的列有 90% 的概率从当天的列子集中选取，10% 的可能从所有可索引列中选取。这样，就可以生成具有内在周期性的数据了。

如图6.1所示的两个列，就是分别常在周二和周日出现的列。它们的到达率存在明显的周期性，需要预测模块进行挖掘和预测。

此外，为了保证查询种类的全面性，测试系统能否平衡各类查询的代价，我还加入了 INSERT、DELETE 等非只读的查询。最终生成的负载中，DELETE 查询占



(a) 常在周二出现的列　　(b) 常在周日出现的列

图 6.1　不同周期的列的到达率折线（时间粒度：天）

约 6.94%，INSERT 查询占约 6.94%，SELECT 查询占约 86.11%。

6.3.2　模块设计

6.3.2.1　聚类模块

本系统中用 *K*-Means 算法依据历史到达率对历史负载查询中涉及到的数据库表列进行分类。具体来说，*K* 的大致枚举范围是根据历史负载的特点人为选定的；而 *K* 的具体取值是通过枚举和多轮重复运行，用轮廓系数法（或肘部法则）自动选取的。

需要注意的是，系统所使用的 *K*-Means 算法使用余弦相似度而不是欧几里得距离作为衡量两个列相似程度的度量。这是因为在实际的工作负载中，具有内在联系的查询的历史到达率折现往往是几何相似的关系（如图6.2所示）。因此，余弦相似度能更好地把握这一规律。

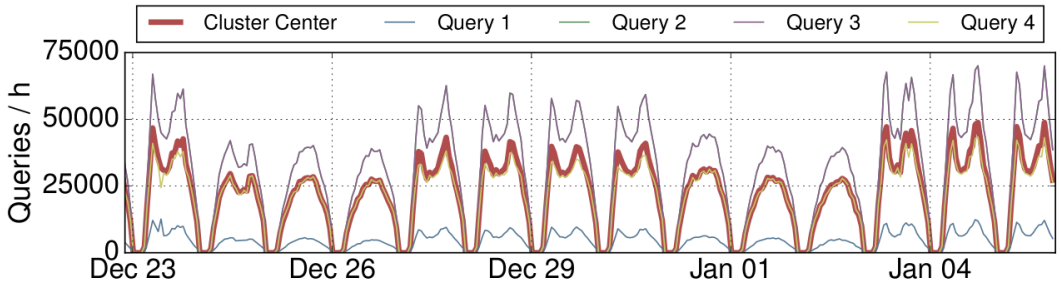


图 6.2　相似的查询到达率曲线是几何相似的

此外，由于系统是将簇内所有列的到达率的平均序列作为列簇的到达率，如果簇内列的到达率折现是几何相似的话，那么这个平均曲线与每个列的折线也都是相似的，于是就可以将预测的列簇折线用等比放缩的方法得到每个列的到达率折线了。从这个需求上讲，余弦相似度同样是更优越的度量方式。

6.3.2.2 预测模块

本模块中，对于聚类算法得到的每个列簇，基于其平均历史到达率序列，各自训练一个 n-gram 的 LSTM 时间序列预测模型 [11]，用于预测该列簇未来的平均到达率。具体来说，对每个列簇，系统将每段连续一段时间粒度 W_h（如一周 7 天）的历史到达率作为输入，其后一个时间粒度的负载作为标签，进行 LSTM 模型训练。预测时，设预测窗口的时间粒度数目为 W_f，由于模型模型每次只预测接下来一个时间粒度的到达率，所以要把预测值作为真实数据值输入模型，并迭代运行模型 W_f 次。

这个 LSTM 模型本身是简单的，如图6.3所示，就是一系列 LSTM 单元构成的隐藏层和一个线性层（Linear）相连，由后者输出结果的结构。

```
PredRNN(
  (lstm): LSTM(1, 100)
  (linear): Linear(in_features=100, out_features=1, bias=True)
)
```

图 6.3 LSTM 时间序列预测模型结构

6.3.2.3 执行模块

本模块利用每列的预测到达率，执行知识图谱负载查询并动态调整索引。为了有效平衡各类查询的性能和存储成本，实验中采用一种启发式的方法 [7]，并设定了索引数目的上限和每轮替换掉的索引数目。

6.3.3 实验过程

实验中使用 Anaconda 搭建 Python 环境，用 MySQL 关系数据库存放数据，用 numpy 和 pandas 等科学计算包完成各种计算，用 PyTorch 搭建和训练 LSTM 序列预测模型。接着按照第 2章中的设计的算法，编写和调试 KGWB 系统代码，设定系统的超参数，将 4 周的历史负载交给系统的预处理模块，记录预处理、聚类和模型训练以及预测的耗时。利用预测的负载运行第 5 周的查询任务，并记录每天的系统查询实际耗时、调整索引的耗时以及整周的总耗时。最后，尝试调整系统部分超参数和部分子算法的选择，分析它们对系统性能的影响。

各模块的细节已在第 2章和本章的6.3.2节详细阐述。这里给出实验所使用的超参数，如表6.2所示。

名称	含义	值
K_LOWER	聚类参数 K 的下界	3
K_UPPER	聚类参数 K 的上界	15
npass	K-Means 算法运行轮数	200
epoch	单个 LSTM 模型的训练轮数	100
W_h	用多少个历史时间粒度预测	7
W_f	需要连续预测多少时间粒度	7
lr	LSTM 模型学习率	0.01
hidden_size	隐藏层 LSTM 单元数目	100
max_index_num	索引数目上限	5
replace_index_num	每天最多替换掉的索引数目	4

表 6.2 超参数表

6.3.4 对比实验

作为 KGWB 系统的对比，我用一个静态算法 STATIC 运行相同的负载，对比它们的效率差别。

STATIC 的运作方式很简单，就是读取前 4 周的历史负载，选择到达率最高的若干列（索引数量上限与 KGWB 系统相同）建立索引，然后用这一套索引执行第 5 周的所有负载，不做任何动态的调整和维护。

6.4 实验结果

6.4.1 聚类模块

聚类模块读取预处理模块生成的如图6.4所示的历史负载三元组集合（三元组依次为时间戳、SQL 查询、可索引列的列表），准确地判断出总共有 7 类可索引列，且效率很高，运行时间不超过 2 秒，几乎可以忽略不计。

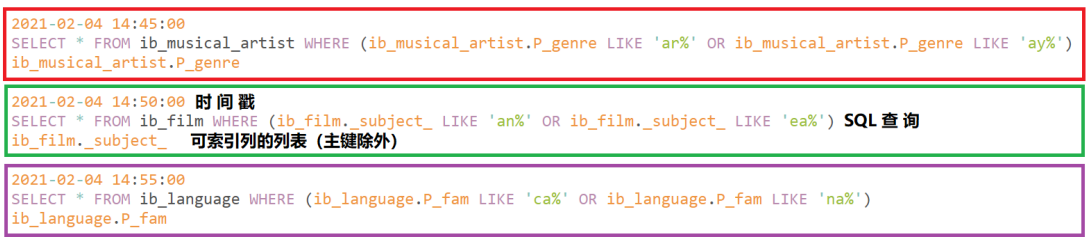


图 6.4 聚类模块输入例

轮廓系数随 K 值的变化折线如图6.5所示，7 个列簇的平均到达率折线图如图6.6所示。

聚类模块的输出是一个 DataFrame 的列表，每个 DataFrame 是一个聚合后的

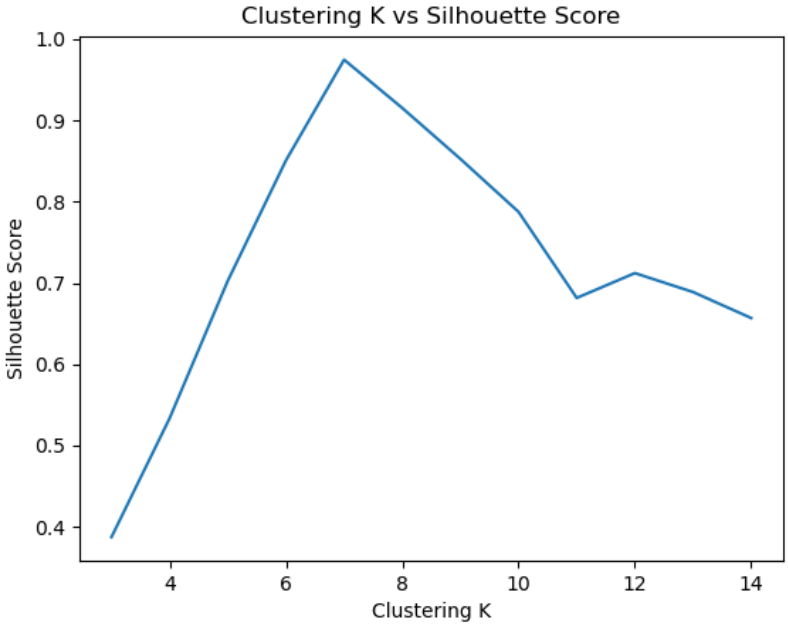


图 6.5　轮廓系数随 *K* 值变化的折线图

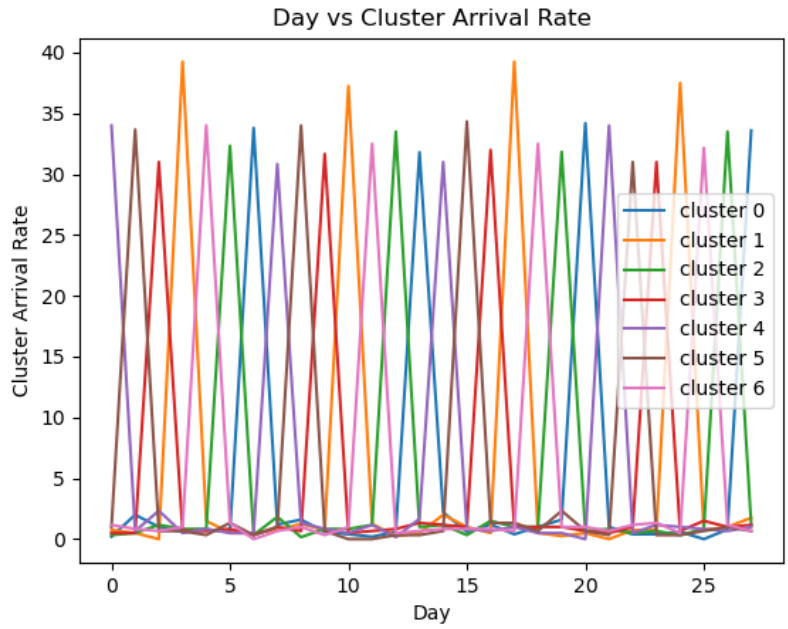


图 6.6　7 个列簇的平均到达率折线

列簇的到达率统计信息，如图6.7所示。

```
[                               0   1   2   3   4   ...  30  31  32  33  34
ib_musical_artist.P_currentMembers  22   1   0   1   1  ...   1   0   0   2   0
ib_person.P_birthPlace              32   3   1   0   2  ...   0   1   1   3   1
ib_album.P_name                     29   2   1   0   3  ...   0   1   2   0   1
ib_album.P_genre                    34   0   3   1   3  ...   1   2   1   0   0
ib_musical_artist.P_name            39   1   1   0   1  ...   2   1   0   1   1

[5 rows x 35 columns],                          0   1   2   3   4   5   ...  29  30  31  32  33  34
ib_musical_artist.P_url     1   0   1   1  22   1  ...   2   0   1  30   1   0
ib_film.P_language          1   2   0   1  34   1  ...   0   0   1  31   0   2
ib_album.P_length           1   2   2   0  32   1  ...   0   0   2  27   0   1
ib_musical_artist.P_genre   3   1   1   0  39   0  ...   1   1   0  32   0   2
ib_language.P_fam           1   0   0   1  26   0  ...   1   1   1  37   0   0

[5 rows x 35 columns],                       0   1   2   3   4   5   6   ...  28  29  30  31  32  33  34
ib_film.P_distributor    0  32   2   1   3   2   5  ...   0  36   1   1   0   0   1
ib_album.P_type          2  33   1   0   1   0   2  ...   1  30   0   2   0   1   0
ib_film.P_name           1  29   2   1   0   0   0  ...   0  35   0   1   1   0   0
ib_person.P_occupation   1  39   2   2   1   0   1  ...   1  28   1   2   0   3   0

[4 rows x 35 columns],                                0   1   2   3   4   5   ...  29  30  31  32  33  34
ib_album.P_cover                 1   1  28   0   3   1  ...   3  33   3   0   2   0
ib_musical_artist.P_yearsActive  2   1  42   0   0   0  ...   1  29   0   0   2   0
ib_album.P_artist                1   1  35   1   0   2  ...   1  32   1   1   0   1
ib_person.P_image                0   2  30   0   0   2  ...   3  30   0   0   0   4
ib_musical_artist.P_background   0   1  28   1   1   0  ...   1  31   1   0   1   1

[5 rows x 35 columns],                          0   1   2   3   4   5   ...  29  30  31  32  33  34
ib_album._subject_          1   0   1  38   0   1  ...   3   0  36   1   0   0
ib_person._subject_         1   0   0  36   0   2  ...   0   0  40   2   1   3
ib_musical_artist.P_label   3   0   0  46   3   1  ...   1   0  37   2   1   0
ib_album.P_reviews          1   0   0  34   0   0  ...   2   0  35   1   2   2
ib_film.P_music             1   2   1  39   0   0  ...   1   2  46   0   0   0
```

图 6.7　聚类模块输出示例（部分）

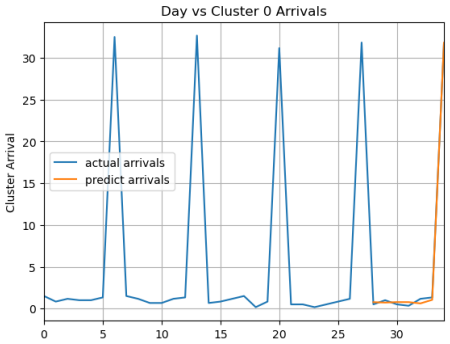
6.4.2　预测模块

预测模块对 7 个列簇分别建立 LSTM 序列预测模型，每个模型训练 100 个 epoch，即能准确地预测出各个列簇的未来到达率走势。重复实验 5 次，平均消耗时间（所有 7 个列簇的模型的训练和未来一周的到达率预测）为 39.57 秒，如表6.3所示（时间单位均为秒）。

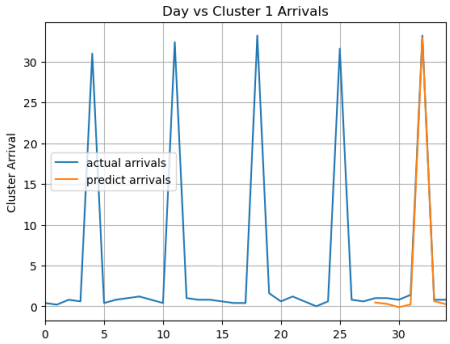
篇幅所限，这里展示 4 个类的预测结果，如图6.8所示。

对每个单列来说，经过缩放，预测效果同样是令人满意的，如图6.9所示。所有列的预测相对误差的平均值约为 3.11%，方差为 0.059。

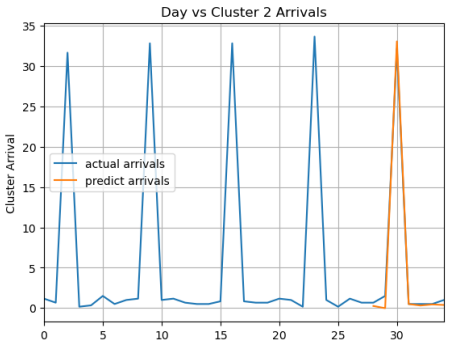
预测模块输出的是每个可索引列第 5 周的预测到达率序列，因此是也是一张 DataFrame，如图6.10所示。



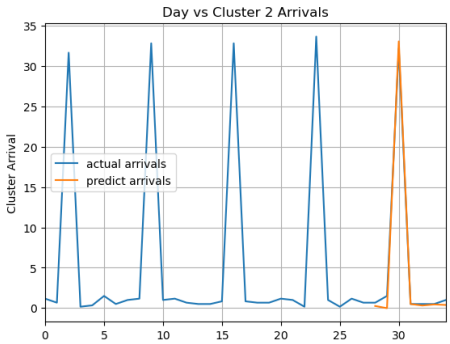
(a) 列簇 0 的到达率预测



(b) 列簇 1 的到达率预测

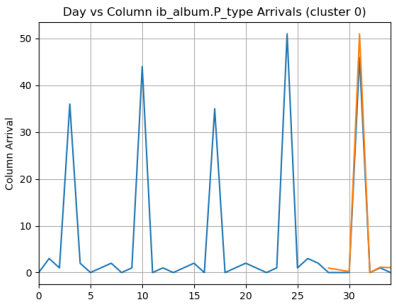


(c) 列簇 2 的到达率预测

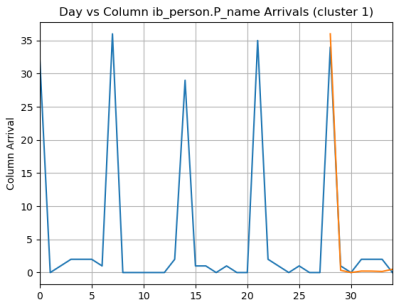


(d) 列簇 3 的到达率预测

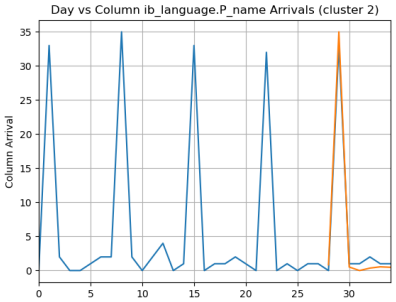
图 6.8 列簇到达率预测结果



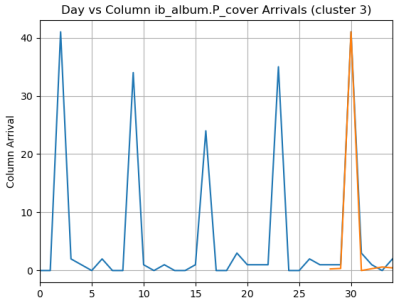
(a) 列簇 0 中一列的到达率预测



(b) 列簇 1 中一列的到达率预测



(c) 列簇 2 中一列的到达率预测



(d) 列簇 3 中一列的到达率预测

图 6.9 单列的到达率预测结果

实验次序	训练时间	预测时间	总时间	平均 loss
1	39.40	0.06	39.46	$< 10^{-3}$
2	39.44	0.06	39.50	$< 10^{-3}$
3	38.23	0.07	38.30	$< 10^{-3}$
4	40.87	0.05	40.92	$< 10^{-3}$
5	39.60	0.06	39.66	$< 10^{-3}$
平均	39.51	0.06	39.57	$< 10^{-3}$

表 6.3 预测模块运行时间

	0	1	2	3	4	5	6
ib_musical_artist._subject_	0.303547	0.530159	0.527587	0.69481	46	0.557559	0
ib_album.P_genre	0.257355	0.449483	0.447302	0.589078	39	0.472713	0
ib_language._subject_	0.217762	0.380331	0.378486	0.498451	33	0.399988	0
ib_film.P_director	0.270552	0.472533	0.470241	0.619287	41	0.496954	0
ib_person.P_name	0.250756	0.437957	0.435833	0.573974	38	0.460592	0
ib_language.P_iso	0	0.336279	48	0.131377	0.071077	0.180949	0.149549
ib_person._subject_	0	0.287239	41	0.112218	0.060711	0.15456	0.127739
ib_film.P_name	0	0.245204	35	0.095796	0.051827	0.131942	0.109046
ib_film.P_distributor	0	0.252209	36	0.098533	0.053307	0.135712	0.112161
ib_album.P_artist	0	0.266221	38	0.104007	0.056269	0.143251	0.118393
ib_musical_artist.P_background	0	0.238198	34	0.093059	0.050346	0.128172	0.10593
ib_album.P_reviews	0.883836	45	0.576521	0.822609	0.697904	0.344306	0
ib_album._subject_	0.707069	36	0.461217	0.658088	0.558323	0.275445	0
ib_album.P_length	0.785632	40	0.512463	0.731208	0.620359	0.30605	0
ib_film.P_language	0.648146	33	0.422782	0.603247	0.511796	0.252491	0
ib_album.P_name	0.707069	36	0.461217	0.658088	0.558323	0.275445	0
ib_person.P_image	0.707069	36	0.461217	0.658088	0.558323	0.275445	0
ib_album.P_producer	0	0.620218	0.706488	0.563523	0.402372	0.74928	38
ib_person.P_birthPlace	0	0.652861	0.743671	0.593182	0.42355	0.788716	40
ib_film._subject_	0	0.603896	0.687896	0.548694	0.391784	0.729562	37
ib_person.P_occupation	0	0.767112	0.873814	0.696989	0.497671	0.926741	47
ib_language.P_fam	0	0.669182	0.762263	0.608012	0.434139	0.808434	41
ib_musical_artist.P_genre	0	0.571253	0.650712	0.519035	0.370606	0.690126	35

图 6.10 预测模块输出示例（部分）

6.4.3 执行模块

相同的 4 周历史负载和 1 周待执行查询，分别交给 KGWB 系统和 STATIC 算法，消耗时间的对比如图6.11所示（这里 KGWB 系统的时间是包含查询执行和索引维护的总时间）。

重复实验，得到如表6.4所示的结果。

从图中和表中可以发现，KGWB 系统的执行效率相比于 STATIC 算法获得了约 25% 的提升；甚至即便加上聚类模块和预测模块的运行时间，还是比 STATIC 算法的耗时更少。

由于最大索引数目的限制，索引占用的空间同样很小。KGWB 执行模块运行过程中，数据库上所有非主键的索引大小平均占用 5.23MB 的空间，相对于数百 MB 的数据集来说几乎可以忽略。也就是说，KGWB 系统通过很小的额外时空代价，就换来了查询负载执行效率的巨大提升。

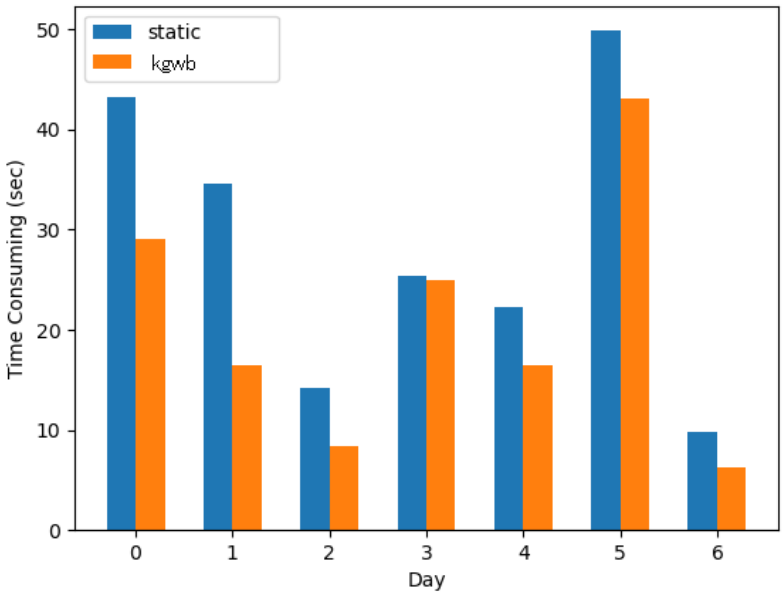


图 6.11 KGWB 系统和 STATIC 算法执行第 5 周查询时间对比

实验次序	STATIC 算法用时	KGWB 执行模块用时	相对提升
1	199.53	149.22	25.20%
2	202.15	150.19	25.70%
3	205.44	150.01	26.98%
4	199.39	148.22	25.66%
5	200.75	152.33	24.12%
平均	201.45	149.99	25.53%

表 6.4 执行模块运行时间

6.5 参数影响分析

在小规模数据下，本系统受超参数影响较小。

对于聚类模块，由于聚类参数 K 的上下界设定比较宽松，所以算法能准确地找到最佳的分类方案。

对于预测模块，实验发现，若减少隐藏层的 LSTM 单元数目、减少训练 epoch，可以一定程度上缩短预测模块的运行时间，但可能无法发掘出更为隐蔽复杂的序列模式。

对于执行模块，若增加最大索引列的数目，会导致 KGWB 系统和 STATIC 算法的差距缩小。这也是自然的，因为每个时间粒度内涉及到的索引数目是有限的，能建立的索引越多，KGWB 系统的动态索引的优势相对也就越小。

第 7 章　相关工作

在传统关系数据库领域，负载预测和索引优化问题受到了广泛关注，也有相当深入的研究。Das 等人提出了一种使用手动构建的规则层次结构的数据库服务自动扩展解决方案 [1]。资源需求估算器从 DBMS 的内部延迟，资源利用率和等待统计信息中得出信号，以确定对资源的需求是高还是低。他们的工作侧重于短期趋势，并单独估计每种资源的需求。这类方法都时估计将来对每种类型资源的需求。而 Lin Ma 等人指出，对查询进行聚类时，衡量语义相似度的效果不如衡量历史负载到达率序列的相似度 [6]。

M. Holze 等人介绍的工作则适用于连续的工作负载分析，专门针对基于流的轻量级操作进行分类，帮助实现可控的质量损失和自我管理 [12]。这种在线分类算法效率稍低，但能够实时地维护聚类结果，在面对连续负载时灵活度更高。

早前还有关于 DBMS 性能建模和诊断的研究。DBSeer 在给定工作负载变化的情况下预测“What-if”问题的答案，例如估算磁盘 I/O 以应对将来的工作负载波动 [3]。该模型基于事务类型对工作负载进行聚类，并根据事务混合预测系统的资源利用率。这个模型假定交易类型固定，是一种离线模型。而本文的工作不仅关注当前的工作负载混合，还可以预测未来的工作负载。

第 8 章　结论

本文提出了一种基于负载预测的知识图谱索引自动动态维护的系统 KGWB，由预处理、聚类、预测和执行四个模块组成。预处理模块解析和统计出历史负载中各可索引列的历史到达率序列，聚类模块用余弦相似度度量对历史负载中的可索引数据库表列进行聚类。接着，预测模块对每个列簇构建 LSTM 时间序列模型来预测各个簇的未来到达率，执行模块基于这个预测进行动态的索引调整。

该系统它针对大规模知识图谱系统的存储和查询特点，相比于静态配置方法大大提升了系统的查询性能，且兼顾了系统性能以及负载预测和索引维护的成本。

参考文献

[1] DAS S, LI F, NARASAYYA V R, et al. Automated Demand-Driven Resource Scaling in Relational Database-as-a-Service[A/OL]. [C], New York, NY, USA : Association for Computing Machinery, 2016 : 1923–1934. https://doi.org/10.1145/2882903.2903733.

[2] Rogers J, Papaemmanouil O, Cetintemel U. A generic auto-provisioning framework for cloud databases[A]. 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)[C], 2010 : 63 – 68.

[3] MOZAFARI B, CURINO C, JINDAL A, et al. Performance and Resource Modeling in Highly-Concurrent OLTP Workloads[A/OL]. SIGMOD '13 : Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data[C], New York, NY, USA : Association for Computing Machinery, 2013 : 301–312. https://doi.org/10.1145/2463676.2467800.

[4] Narayanan D, Thereska E, Ailamaki A. Continuous resource monitoring for self-predicting DBMS[A]. 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems[C], 2005 : 239 – 248.

[5] SALZA S, TERRANOVA M. Workload Modeling for Relational Database Systems[M/OL] // DEWITT D J, BORAL H. Database Machines: Fourth International Workshop Grand Bahama Island, March 1985. New York, NY : Springer New York, 1985 : 233 – 255. https://doi.org/10.1007/978-1-4612-5144-6_12.

[6] MA L, VAN AKEN D, HEFNY A, et al. Query-Based Workload Forecasting for Self-Driving Database Management Systems[A/OL]. [C], New York, NY, USA : Association for Computing Machinery, 2018 : 631–645. https://doi.org/10.1145/3183713.3196908.

[7] CHAUDHURI S, NARASAYYA V R. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server[A]. VLDB '97 : Proceedings of the 23rd International Conference on Very Large Data Bases[C], San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1997 : 146–155.

[8] ELMAN J L. Finding structure in time[J]. Cognitive science, 1990, 14(2): 179 – 211.

[9] HOCHREITER S, SCHMIDHUBER J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735 – 1780.

[10] LEHMANN J, ISELE R, JAKOB M, et al. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia[J]. Semantic web, 2015, 6(2): 167 – 195.

[11] HOLZE M, RITTER N. Autonomic Databases: Detection of Workload Shifts with n-Gram-Models[A]. ATZENI P, CAPLINSKAS A, JAAKKOLA H. Advances in Databases and Information Systems[C], Berlin, Heidelberg: Springer Berlin Heidelberg, 2008: 127 – 142.

[12] HOLZE M, GAIDIES C, RITTER N. Consistent On-Line Classification of Dbs Workload Events[A/OL]. CIKM '09: Proceedings of the 18th ACM Conference on Information and Knowledge Management[C], New York, NY, USA: Association for Computing Machinery, 2009: 1641–1644. https://doi.org/10.1145/1645953.1646193.

[13] 孙路明, 张少敏, 姬涛, et al. 人工智能赋能的数据管理技术研究 [J]. 软件学报, 2020, 000(003): 600 – 619.

[14] 王鑫, 邹磊, 王朝坤, et al. 知识图谱数据管理研究综述 [J]. 软件学报, 2019(7): 2139 – 2174.

[15] WILKINSON K. Jena property table implementation[J], 2006.

[16] Holze M, Haschimi A, Ritter N. Towards workload-aware self-management: Predicting significant workload shifts[A]. 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)[C], 2010: 111 – 116.

[17] Kanungo T, Mount D M, Netanyahu N S, et al. An efficient k-means clustering algorithm: analysis and implementation[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 24(7): 881 – 892.

[18] PHAM D T, DIMOV S S, NGUYEN C D. Selection of K in K-means clustering[J]. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 2005, 219(1): 103 – 119.

[19] DOWNEY C, HEFNY A, BOOTS B, et al. Predictive state recurrent neural networks[A]. Advances in Neural Information Processing Systems[C], 2017 : 6053 – 6064.

[20] GHOSH A, PARIKH J, SENGAR V S, et al. Plan selection based on query clustering[A]. VLDB'02: Proceedings of the 28th International Conference on Very Large Databases[C], 2002 : 179 – 190.

[21] GANAPATHI A, KUNO H, DAYAL U, et al. Predicting multiple metrics for queries: Better decisions enabled by machine learning[A]. 2009 IEEE 25th International Conference on Data Engineering[C], 2009 : 592 – 603.

[22] DU N, YE X, WANG J. Towards workflow-driven database system workload modeling[A]. Proceedings of the Second International Workshop on Testing Database Systems[C], 2009 : 1 – 6.