

PageRank + Latent Dirichlet Allocation

1. Page Rank 原理：

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

基本假设：

- **重要性传递：**一个页面的重要性可以通过其链接到的页面来传递。如果一个重要的页面链接到另一个页面，那么被链接的页面也被视为重要。
 - **累积效应：**通常，重要或者高质量的内容会吸引更多的用户关注和引用。这种累积效应导致重要网站获得更多的外部链接，从而在 PageRank 算法中获得更高的评分。
- **链接数量的影响：**一个页面的 PageRank 取决于链接到它的其他页面的数量和质量。链接自许多高质量页面的页面通常会有较高的 PageRank。
 - **网络信任与权威性传递：**如果一个被广泛认为重要或者有用的网站链接到另一个网站，这种行为类似于对被链接网站的一种推荐或认可。这种推荐在互联网上表现为链接，意味着链接来源网站的信任和权威性在一定程度上传递给了链接目标网站。
 - **例：**新闻网站可能会链接到原始报告或相关文章以增加报道的可信度。学术网站可能链接到研究论文或数据库以提供更多背景资料和支持数据。
 - **自我增强循环：**随着网站通过链接获得更高的 PageRank，它在搜索引擎结果中的排名也会提升，这又使得更多的用户能够发现并链接到该网站，进一步增强其重要性。
- **随机浏览者模型：**PageRank 假设网络中的用户是随机点击链接的浏览者。用户在浏览时可能会随机选择任意一个链接进行跳转，或者开始新的搜索。

为什么被链接越多的网站，重要性/相关性越高？

1. **链接推荐**：链接是对目标网站的推荐。多网站链接表示内容价值、可信度和权威性。
2. **社交验证**：网站被多链接传递信任和质量的信号，增加可信度和吸引力。
3. **网络中心性**：链接多的网站在网络中中心性高，内容更相关和重要。
4. **质量累积**：高质量内容产生外部链接，累积为内容和相关性的指标。
5. **信息丰富性**：内容丰富、深度信息的网站吸引各种链接，与网站相关性相关，满足用户需求。

2. 算法概览

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, called “iterations”, through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value.

1. **概率分布**：PageRank 算法的输出是一个概率分布，这个分布代表一个人随机点击链接最终到达每个页面的概率。这意味着每个网页的 PageRank 值可以被视为在任何给定时刻随机访问该页面的可能性。
2. **初始分配**：在计算开始时，通常假设这个概率分布在所有文档中均匀分布。也就是说，每个页面初始的 PageRank 值是相等的，这是一个简化的假设，使得算法从一个中性的状态开始迭代。
3. **迭代计算**：PageRank 值的计算需要通过多次迭代来逐步调整。每次迭代都会根据页面之间的链接结构调整每个页面的 PageRank 值。具体来说，一个页面的 PageRank 值由链接到它的其他页面的 PageRank 值决定，这些值通过链接传递一部分给目标页面。
4. **接近真实值**：虽然初始的 PageRank 值是均匀分布的，但通过迭代过程，这些值会逐渐调整，以更接近理论上的“真实”值。每次迭代后，系统都会更准确地反映页面的实际重要性，这是通过网络中的链接模式和结构来决定的。
5. **适用性**：PageRank 算法不仅可以应用于网页，还可以用于任何大小的文档集合。这使得它在多种不同的信息检索和网络分析场景中都非常有用。

3. 算法简化过程(未考虑阻尼系数)

这里简要解释了如何通过迭代方法计算四个页面（A, B, C, D）的 PageRank 值，实际建议考虑阻尼系数

1. 初始化:

- 每个页面初始的 PageRank 值设为 0.25，因为总共有四个页面，每个页面分配到的 PageRank 总和为 1（ $1 / 4 = 0.25$ ）。

2. PageRank 传递:

- 每个页面在迭代过程中会将其 PageRank 值平均分配给它指向的其他页面。例如，如果页面 B 链接到页面 C 和 A，则 B 将其 PageRank 值的一半（0.125）传递给 C 和 A。

3. 计算例子:

迭代一:

- B 有链接到 C 和 A，因此将一半的 PageRank 传递给 C 和 A。

$$PR(A)' = \frac{PR(B)}{2} = 0.125$$

$$PR(C)' = \frac{PR(B)}{2} = 0.125$$

- C 只有链接到 A，所以将其全部 PageRank 传递给 A。

$$PR(A)' = \frac{PR(B)}{2} + PR(C) = 0.125 + 0.25 = 0.375$$

- D 有链接到所有页面，所以将其 PageRank 平均分为三份，每份大约 0.083，传递给 A。

$$PR(A)' = \frac{PR(B)}{2} + PR(C) + \frac{PR(D)}{3} = 0.125 + 0.25 + 0.083 = 0.458$$

$$PR(B)' = \frac{PR(D)}{3} = 0.083$$

$$PR(C)' = \frac{PR(B)}{2} + \frac{PR(D)}{3} = 0.125 + 0.083 = 0.208$$

- 迭代完成后，A 的新 PageRank 值是各个传入值的总和

$$PR(A)' = \frac{PR(B)}{2} + PR(C) + \frac{PR(D)}{3}$$

在 PageRank 算法中，确保每个页面分配到的 PageRank 总和始终为 1 是非常重要的。这是算法的一个基本特征，用来表示在任何时刻，一个随机网页浏览者处于网络中任何一个页面的概率总和是 100%。

因此，尽管每次迭代中页面的 PageRank 值是根据当前的网络链接状态重新计算的，算法的设计要确保在任何时候 PageRank 的总和都保持为 1，无论是否考虑阻尼因子。因此，由于 A 页面未链接到任意一个其他页面，它将丢失本身的 $PR(A) = 0.25$ ，泄露的 PageRank 值需要重新分配给所有页面。

4. 公式总结：

- 一般情况下，任何页面 u 的 PageRank 值 $PR(u)$ 可以通过所有指向 u 的页面 v 的 PageRank 值除以 v 的出链接数 $L(v)$ 的和来计算：

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

- 其中， B_u 是所有链接到页面 u 的页面集合， $L(v)$ 是页面 v 的出链接数

4. 标准 PageRank Algorithm

1. 计算公式

$$PR(u) = \frac{1 - \alpha}{N} + \alpha \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

- 第一项 $\frac{1 - \alpha}{N}$ 确保了每个页面至少会获得一小部分 PageRank，这一部分对所有页面平均分配，其目的是模拟用户随机跳转到任一页面的行为。
- 第二项是从其他页面通过直接链接得到的 PageRank。
- 参数说明：**
 - $PR(u)$ 是页面 u 的 PageRank 值。
 - N 是网络中的页面总数。
 - α 是阻尼系数，通常设置为 0.85。
 - B_u 是所有直接链接到页面 u 的页面集合。
 - $L(v)$ 是页面 v 的出链接数。
 - $\frac{1 - \alpha}{N}$ 是每个页面基于随机跳转获取的 PageRank 的一部分。

2. 阻尼系数 (Damping factor)

- 阻尼系数（通常表示为 α ，常设为 0.85）用于模拟一个人在随机浏览网页时不仅仅是跟随链接，也有一定的概率随机跳转到网络中的任何其他页面。这个参数有助于解决一些问题，比如没有外部链接的页面（"死胡同"页面）或页面间链接循环的情况。
- 为什么需要阻尼系数？
 - 如果不考虑阻尼系数（即将阻尼系数设为 1），PageRank 的算法会变成一个完全基于链接结构的简单迭代模型。在这种情况下，每个页面会将其全部 PageRank 值均分给所有出站链接的页面。如果一个页面没有出站链接，则它在每次迭代中会将其 PageRank 值“损失掉”，因为没有页面可以接收这部分值。这可以导致几个潜在的问题：
 - a. **排泄页面（Sink Pages）**：排泄页面是指那些没有出站链接的页面。在没有阻尼系数的情况下，这些页面会在每次迭代中接收 PageRank，但不会将 PageRank 传递出去，从而导致 PageRank 在这些页面上累积。随着迭代次数增加，网络中的总 PageRank 会逐渐被这些排泄页面吸收，导致其他页面的 PageRank 值逐渐减小至零。
 - b. **循环结构和孤立集合**：在网络中可能存在循环结构，例如页面 A 链接到页面 B，页面 B 又链接回页面 A。在这种情况下，PageRank 将在循环结构中的页面之间循环流动，而不会逃逸到循环之外。类似地，如果一个子集的页面只相互链接，并且没有外部页面的链接，它们的 PageRank 将在子集内循环，不会流向子集之外。
 - c. **PageRank 泄露**：如果一个页面仅链接到排泄页面，其 PageRank 将最终传递给排泄页面，并且无法从其他任何页面接收 PageRank。随着时间的推移，这将导致非排泄页面的 PageRank 逐渐流失，最终可能导致大多数 PageRank 聚集在少数排泄页面上。
 - d. **PageRank 值的稳定性和收敛性**：不使用阻尼系数的 PageRank 算法可能难以收敛到稳定的状态，特别是在存在复杂链接结构的大规模网络中。阻尼系数有助于保证算法的收敛，因为它引入了一个随机浏览的概念，即有一定比例的 PageRank 是均匀分布的，而不仅仅依赖于链接结构。

3. 马尔科夫链（Markov Chain）

可以近似的将 PageRank 算法近似理解为马尔科夫链，即网页的重要性由其能够从其他网页“吸引”随机游走者的能力决定。

- **状态和转移：**

- **状态：**每个网页被视为一个状态。在马尔可夫链中，这些状态相当于系统可能处于的各种不同的配置或位置。
- **转移：**网页之间的链接定义了状态之间的转移。如果一个网页 A 有一个链接指向网页 B，那么在马尔可夫链模型中，存在一个从状态 A（即网页 A）到状态 B（即网页 B）的转移。
- **转移的概率：**
 - 在 PageRank 算法中，一个页面到其他页面的转移概率由链接结构决定。如果一个页面有多个出站链接，到每个链接页面的转移概率是不同的，取决于该页面的出站链接总数。

4. 计算方法

a. 迭代方法 (Iterative Method)

- i. **初始设置：**设定初始 PageRank 值。通常，每个页面的初始 PageRank 值设为相等，假设网络中有 N 个页面，则每个页面的初始 PageRank 值设为 $\frac{1}{N}$

- ii. **迭代公式：**

$$PR(p_i; t + 1) = \frac{1 - \alpha}{N} + \alpha \sum_{p_j \in M(p_i)} \frac{PR(p_j; t)}{L(p_j)}$$

- p_j 是指向页面 i 的所有页面的集合 $M(p_i)$ 。
- $PR(p_j; t)$ 是页面 j 在时间 t 的 PageRank 值。
- $L(p_j)$ 是页面 j 的出链接数。
- α 是阻尼因子，通常设为 0.85，代表从其他页面通过链接传递来的 PageRank 占总 PageRank 的比例。
- $\frac{1 - \alpha}{N}$ 是每个页面因随机浏览（即用户不通过链接，而是随机选择一个页面）而获得的 PageRank 值。

- iii. **矩阵表示：**

$$R(t + 1) = \alpha MR(t) + \left(\frac{1 - \alpha}{N}\right)E$$

$$R(t) = \begin{bmatrix} PR(p_1; t) \\ PR(p_2; t) \\ \vdots \\ PR(p_i; t) \end{bmatrix}$$

- $R(t)$ 是一个向量，其第 i 个元素代表页面 i 在时间 t 的 PageRank 值。
- M 是转移概率矩阵，其中 $M_{ij} = \frac{1}{L(p_j)}$ 如果页面 j 链接到页面 i ，否则为 0。
- E 是一个所有元素都为 1 的向量，用于表示每个页面因随机浏览得到的基础 PageRank 值。

iv. 迭代过程：

迭代过程持续进行，直到 $\|R(t+1) - R(t)\| < \epsilon$ (预设阈值 ϵ)，此时认为 PageRank 值已收敛。

b. 幂法 (Power Method)

特别适用于找到网络链接矩阵的主要特征向量

PageRank 的计算本质上是利用 Power Method 来估算 Google 矩阵（或称为 PageRank 矩阵）的主特征向量。

i. 初始设置

首先，需要一个初始的概率向量 $x(0)$ ，通常可以选择每个元素等于 $\frac{1}{N}$ ，其中 N 是页面总数。这个向量表示每个页面的初始重要性或 PageRank 值。

ii. 基本迭代公式

$$x(t+1) = Ax(t)$$

- A 是转移概率矩阵，适当调整以应用 PageRank 的阻尼系数
- $x(t)$ 是在迭代步骤 t 中每个页面的 PageRank 向量

为了适用于 PageRank，矩阵 A 由以下方式定义：

$$A = \alpha M + \left(\frac{1-\alpha}{N}\right)E$$

- M 是基于网页链接结构的标准化转移概率矩阵。如果页面 j 链接到页面 i ，则 $M_{ij} = \frac{1}{L(p_j)}$ ；如果没有链接，则为 0。

- α 是阻尼因子，通常设置为 0.85，表示 85% 的时间用户会通过链接跟随到下一个页面，15% 的时间会随机跳转到网络中的任一页面。
- E 是一个所有元素都为 1 的矩阵，用于添加随机跳转的概率。

iii. 收敛条件

迭代持续进行，直到向量 $x(t)$ 收敛，即 $\|x(t+1) - x(t)\| < \epsilon$

c. 不同计算方法的应用场景选择

- **PageRank 计算：**由于 PageRank 的定义就是链接矩阵的主特征向量，因此 Power Method 是理想的选择。它直接针对 PageRank 的核心数学定义进行优化，效率高，实现简单。
- **其他类型的问题：**如果问题涉及求解一般的线性方程组或需要找到多个特征值和特征向量，可能会选择使用更通用的迭代方法，如 Jacobi、Gauss-Seidel 或其他数值方法。

5. PageRank 在文档搜索召回中的应用

PageRank 在文档数据库召回中的适用性

a. 文档间的引用关系：

如果文档数据库中的文档相互之间存在引用关系（如学术论文、法律文件、技术文档等），这些引用可以被视为链接。在这种情况下，PageRank 可以帮助确定哪些文档因为被频繁引用而具有较高的重要性或权威性。

b. 非链接元数据的整合：

对于那些不具有明显引用关系的文档，PageRank 算法需要适当修改或与其他技术（如文本相似度、元数据分析等）结合使用，以评估和排序文档的相关性。

PageRank 的潜在优势

a. 提高重要文档的可见性：

使用 PageRank 可以帮助用户在大量搜索结果中更快地找到那些具有高度影响力或核心地位的文档。

b. 利用文档间的关系：

在某些类型的数据库中，文档间的引用关系可能会提供重要的上下文信息，PageRank 通过这些关系挖掘出可能对用户查询特别相关的文档。

PageRank 的限制

a. 缺乏文档内部内容分析:

PageRank 算法本身不处理文档的内容。对于依赖内容匹配的召回系统, 仅使用 PageRank 可能不足以实现高效的召回, 因为它不考虑查询和文档内容之间的匹配度。

b. 更新和维护成本:

如果文档库频繁更新, 维护文档间引用关系的 PageRank 计算可能会变得复杂和资源密集。

6. 如何将 PageRank 与当前召回排序规则结合

1. 目前召回排序遵循的规则:

■ 打开时间:

$$f(\text{score}) = \begin{cases} 160 - (\text{diffSeconds} / 8640) & 0 \leq \text{diffDays} < 2 \\ 160 - 5 * \text{diffDays} - (\text{diffSeconds} / 8640) & 2 \leq \text{diffDays} < 4 \\ 140 - (\text{diffSeconds} / 8640) & 4 \leq \text{diffDays} < 7 \\ 10 + 50 * \exp(-0.05 \text{diffDays}) & 7 < \text{diffDays} \leq 15 \\ 10 + 50 * \exp(-0.05 \text{diffDays}) & 15 < \text{diffDays} \leq 30 \\ 10 + 12 * \exp(-0.005 \text{diffDays}) & \text{diffDays} > 30 \end{cases}$$

■ 编辑时间:

$$f(\text{score}) = \begin{cases} 180 - (\text{diffSeconds} / 8640) & 0 \leq \text{diffDays} < 2 \\ 180 - 5 * \text{diffDays} - (\text{diffSeconds} / 8640) & 2 \leq \text{diffDays} < 4 \\ 160 - (\text{diffSeconds} / 8640) & 4 \leq \text{diffDays} < 7 \\ 20 + 50 * \exp(-0.04 \text{diffDays}) & 7 < \text{diffDays} \leq 15 \\ 15 + 50 * \exp(-0.04 \text{diffDays}) & 15 < \text{diffDays} \leq 30 \\ 22 * \exp(-0.001 \text{diffDays}) & \text{diffDays} > 30 \end{cases}$$

■ 打开 / 编辑次数: $f(\text{score}) = q_t * \log(p_t * \text{openCount})$ (q 和 p 的值与时间 t 有关)

■ 星标, 点赞, 评论: 固定至常用 +12 分 星标 +10 分 点赞 + 2 分 评论 + 5 分 活跃圈子 +5 分

■ 文件名匹配 +100 文件名拼音匹配 +50

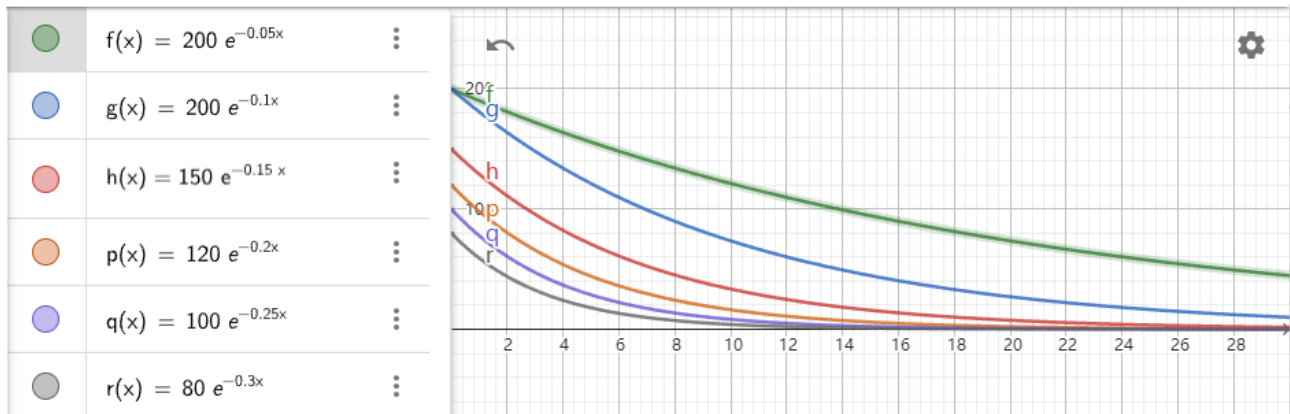
2. 拟合当前评价公式

打开 / 编辑时间:

$$f(\text{score}) = A \cdot \exp(-\lambda \cdot \text{diffdays})$$

- $0 \leq \text{diffdays} < 2$: $A=200$, $\lambda=0.05$ (高初始得分, 低衰减率, 突出近期的重要性)
- $2 \leq \text{diffdays} < 4$: $A=200$, $\lambda=0.10$ (适中初始得分, 适度衰减, 仍保持较高重要性)

- $4 \leq \text{diffdays} < 7$: $A=150, \lambda=0.15$ (较低初始得分, 较快衰减, 反映一周内逐渐减少的相关性)
- $7 \leq \text{diffdays} < 15$: $A=120, \lambda=0.20$ (初始得分进一步降低, 衰减加速)
- $15 \leq \text{diffdays} < 30$: $A=100, \lambda=0.25$ (较低的初始得分和较高的衰减率, 用于较长时间范围)
- $\text{diffdays} \geq 30$: $A=80, \lambda=0.30$ (最低的初始得分和最快的衰减, 用于处理较旧的文档)



打开 / 编辑次数:

$$w_i = e^{-\lambda(t-t_i)}$$

$$N' = \sum_{i=1}^n w_i$$

$$f(\text{score}) = q * \log(p \times N')$$

- λ 为衰减系数, t_i 是第*i*次打开 / 编辑的时间, 当前时间为*t*, w_i 为每次打开 / 编辑的权重
- N' 为所有权重的和
- 距离当前时间越远的访问对总评分的贡献应当越小。

总评分公式:

$$\text{Total Score} = w_1 \times \text{PageRank Score} + w_2 \times \text{Time-Based Score} + w_3 \times \text{Static Factors Score}$$

$$= \omega_1 \times PageRankScore + \omega_2 \times (q * \log(p \times N') + A \cdot \exp(-\lambda \cdot diffdays)) \\ + \omega_3 \times (\text{星标} + \text{固定常用} + \text{文件名匹配} + \dots)$$

7. 如何获取 PageRank Score

a. 文本主题建模

利用 LDA 对文档进行主题建模，得到文档 - 主题分布以及主题 - 单词分布，同时将候选关键词化分到不同的主题中，并得到候选关键词和主题的相关程度。定义关键词 r_i 和主题 Z 的关系 $c(r_i|z)$

$$c(r_i|z) = \frac{\sum_{w \in r_i} p(w|z)}{N(r_i)}$$

b. 基于短语的主题权重 PageRank 算法

(<http://pasa-bigdata.nju.edu.cn/achievement/KeyExtract.html>) <--- 由此启发

两个阶段：构建权重短语图和运行主题权重 PageRank 算法。

当 LDA 主题模型训练完成之后，同时通过 LDA 模型训练可以得到文档的主题分布 $p(z|d)$ 。定义候选关键词之间的距离如公式：

$$dist(r_i, r_j) = \sum_{d_i \in pos(r_i)} \sum_{d_j \in pos(r_j)} \frac{1}{|d_i - d_j|}$$

其中， $pos(r_i)$ 表示短语 r_i 在文档中出现的位置。在每个文档的相关主题上，由候选关键词在每个主题的初始权重和候选关键词之间的近似距离本文可以构建主题的权重短语图，完成基于短语的主题权重 PageRank 第一阶段工作。

权重短语图构建之后，下一阶段是运行主题权重 PageRank 算法。本文的主题权重 PageRank 算法将候选关键词和主题的相关程度作为其初始 PR 值。同时定义顶点 r_i 在主题 Z 下的 PR 值计算公式如下所示：

$$PR_z(r_i) = (1 - \lambda)c_z(r_i) + \lambda \sum_{r_j \in s(r_i)} \frac{dist(r_i, r_j)}{O(r_j)} PR_z(r_j)$$

其中, $c_z(r_i)$ 表示候选关键词 r_i 的初始权重值, $s(r_i)$ 表示指向顶点 r_i 的顶点集合, $O(r_j)$ 表示顶点 r_j 的出度, 根据上述公式算法可以得到主题相关的候选关键词 r_i 在此主题下的重要性打分 $PR_z(r_i)$ 。由此完成算法的第二阶段, 得到每个主题下候选关键词的打分。

8. LDA, PageRank 与搜索引擎召回排序结合

💡 大多数文档之间并没有网页之间的频繁链接（引用），因此使用文档之间的主题距离作为 pagerank 链接图中边的权重来表示文档之间的链接强度或相似性

1. 文档主题建模：

首先, 对文档集进行 LDA 主题建模, 确定每个文档的主题分布。这个步骤涉及以下公式和计算:

- 对于每个文档 d , 利用 LDA 模型计算其在每个主题 z 下的条件概率 $p(z | d)$ 。这表示文档 d 属于主题 z 的概率。

$$p(\text{词语}|\text{文档}) = \sum_{\text{主题}} p(\text{词语}|\text{主题}) \times p(\text{主题}|\text{文档})$$

2. 计算文档之间的主题距离：

利用文档的主题分布, 计算文档间的主题距离, 这将用于调整 PageRank 值。文档间的主题距离可以通过以下公式计算:

- 对于两个文档 r_i 和 r_j , 计算他们之间的主题距离 $dist(r_i, r_j)$:

$$dist(r_i, r_j) = \sum_z |p(z|r_i) - p(z|r_j)|$$

这里的 $dist$ 衡量了两个文档在主题分布上的差异。

3. 结合主题距离的 PageRank 计算

使用文档间的主题距离调整传统的 PageRank 计算。传统 PageRank 的更新公式通常是:

$$PR(r) = \frac{1-a}{N} + a \cdot \sum_{r' \in B_r} \frac{PR(r')}{L(r')}$$

其中， α 是阻尼系数， N 是总文档数， B_r 是指向文档 r 的其他文档集合， $L(r')$ 是文档 r' 的出链数目。

在结合主题距离后，PageRank 的更新可以考虑文档间的主题相似性或距离：

首先，将距离转换为相似度：

$$S_{r_i r_j} = 1 - \frac{dist(r_i, r_j)}{\max(dist(r_i, r_j))}$$

其中 $\max(dist(r_i, r_j))$ 是所有文档对之间主题距离的最大值，用于归一化，确保 $S_{r_i r_j}$ 在 0 到 1 之间。

接着，使用转换后的相似度矩阵 S 来定义文档间的链接权重。

$$PR_i^{(t+1)} = \frac{1 - \alpha}{N} + \alpha \cdot \sum_{r' \in L_i} \frac{PR_j^{(t)} \cdot S_{r_i r_j}}{|L_j|}$$

其中：

- $PR_i^{(t+1)}$ 是文档 i 在迭代 $t+1$ 时的 PageRank 值
- N 是文档总数
- $|L_j|$ 是链接出文档 j 的文档总数
- $S_{r_i r_j}$ 是文档 j 到 i 的相似度

4. 结合点击率 (CTR) 分析

使用点击率来调整 PageRank 的计算，以此反映文档的实际吸引力。

数学模型调整：

$$PR_i^{(t+1)} = \frac{1 - \alpha}{N} + \alpha \cdot (\beta \cdot \sum_{r' \in L_i} \frac{PR_j^{(t)} \cdot S_{r_i r_j}}{|L_j|} + (1 - \beta) \cdot CTR_i)$$

其中：

- β 是一个调节参数，用于平衡由链接导致的 PageRank 贡献和点击率的影响
- CTR_i 是文档 i 的点击率

9. PageRank 代码封装以及基于 gRPC 的 Golang 调用

有了 gRPC，我们可以一次性的在一个 .proto 文件中定义服务并使用任何支持它的语言去实现客户端和服务端，反过来，它们可以在各种环境中，从 Google 的服务器到你自己的平板电脑——gRPC 帮你解决了不同语言及环境间通信的复杂性。使用 protocol buffers 还能获得其他好处，包括高效的序列化，简单的 IDL 以及容易进行接口更新。

为了将 PageRank 代码封装成一个服务以支持 Golang 语言通过 gRPC 对其进行调用，需要完成以下步骤

步骤 1: 定义 gRPC 服务接口

首先，你需要定义一个 `.proto` 文件，用于描述你的服务接口和消息格式。这个定义将包括你想要通过 gRPC 提供的方法，例如，获取文档排名。

创建 `document_service.proto` 文件：

Python

```
1 syntax = "proto3";
2
3 package documentprocessor;
4
5 option go_package = "package/proto;proto";
6
7 // 定义服务消息
8 message DocumentRequest {
9     string directory = 1;
10    repeated int32 clicks = 2; // 添加点击次数数组
11 }
12
13 message DocumentResponse {
14     string documentName = 1;
15     double pageRankScore = 2;
16 }
17
18 message DocumentList {
19     repeated DocumentResponse documents = 1;
20 }
21
22 // 定义服务接口
23 service DocumentProcessor {
24     rpc ProcessDocuments (DocumentRequest) returns (DocumentList);
25 }
```

步骤 2：安装必要的库

- 对于 Python：你需要使用 `grpcio-tools` 生成服务代码。

Bash

```
1 pip install grpcio grpcio-tools
```

- 对于 Go：安装 Go gRPC 库和协议编译器插件：

Bash

```
1 go install google.golang.org/grpc/cmd/protoc-gen-go-grpc@latest
2 go install google.golang.org/protobuf/cmd/protoc-gen-go@latest
```

步骤 3 : 生成 Python gRPC 代码

使用 `grpc_tools.protoc` 来生成 Python 的 gRPC 代码：

Bash

```
1 python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. document_service.proto
```

这将生成包含服务类和存根的 Python 文件，你可以用它们来实现服务和创建客户端。

步骤 4: 实现 gRPC 服务

在 Python 中实现 gRPC 服务。创建一个文件 `document_service.py`，并实现你在 `.proto` 文件中定义的服务：

Python

```
1 import grpc
2 from concurrent import futures
3 import document_service_pb2
4 import document_service_pb2_grpc
5
6 # 这里为先前的文档处理代码def
7
8 class DocumentProcessorServicer(document_service_pb2_grpc.DocumentProcessorServicer):
9     def ProcessDocuments(self, request, context):
10         directory = request.directory
11         documents, document_names = process_documents(directory)
12         dictionary, corpus = prepare_corpus(documents)
13         lda = lda_model(corpus, dictionary, num_topics=5)
14         doc_topics = get_topic_distribution(lda, corpus)
15         distances = calculate_all_topic_distances(doc_topics)
16         similarities, links = convert_distance_to_similarity(distances)
17         clicks = np.array(request.clicks) # 使用从客户端接收的点击数组
18         max_clicks = np.max(clicks)
19         ctr = clicks / max_clicks
20         pr_with_ctr = page_rank_with_ctr(links, similarities, ctr)
21         doc_pagerank = list(zip(document_names, pr_with_ctr))
22         sorted_doc_pagerank = sorted(doc_pagerank, key=lambda x: x
[1], reverse=True)
23
24         response_list = document_service_pb2.DocumentList()
25         for doc_name, pr_score in sorted_doc_pagerank:
26             response_list.documents.add(documentName=doc_name, pageRank
Score=pr_score)
27
28         return response_list
29
30
31 def serve():
32     server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
33     document_service_pb2_grpc.add_DocumentProcessorServicer_to_server(D
ocumentProcessorServicer(), server)
34     server.add_insecure_port('[::]:50051')
```

```
35     print("gRPC starting")
36     server.start()
37     server.wait_for_termination()
38
39 if __name__ == '__main__':
40     serve()
41
```

步骤 5: 生成 Go 客户端代码

将上述 `document_service.proto` 复制到 Go 项目文件下后执行以下命令来生成 Go 代码

Bash

```
1 protoc -I . --go_out=. --go-grpc_out=. document_service.proto
```

步骤 6 : 创建 Go gRPC 客户端

创建 `document_client.go` 以调用 gRPC 服务

Go

```
1 package main
2
3 import (
4     "context"
5     "log"
6     "time"
7
8     "google.golang.org/grpc"
9     pb "path/to/your/generated/proto/package" // 将此路径修改为步骤5生成的
    包路径
10 )
11
12 func main() {
13     // 设置服务器地址和端口
14     address := "localhost:50051"
15     conn, err := grpc.Dial(address, grpc.WithInsecure(), grpc.WithBlock
    ())
16     if err != nil {
17         log.Fatalf("did not connect: %v", err)
18     }
19     defer conn.Close()
20
21     c := pb.NewDocumentProcessorClient(conn)
22
23     // 调整 timeout 时间
24     ctx, cancel := context.WithTimeout(context.Background(), time.Secon
    d)
25     defer cancel()
26
27     // 准备请求数据
28     directory := "C:/pythonProject/PageRank/技术1"
29     clicks := []int32{5, 10, 15, 20, 25, 5, 10, 15, 20, 25, 5, 10, 1
    5, 20, 25, 5, 10, 15, 20, 25}
30     //以上自行输入每个文件的点击量
31
32     // 发送请求并接收响应
33     r, err := c.ProcessDocuments(ctx, &pb.DocumentRequest{
34         Directory: directory,
```

```

35         Clicks:    clicks,
36     })
37     if err != nil {
38         log.Fatalf("could not process documents: %v", err)
39     }
40
41     // 输出从服务端接收的数据
42     for _, doc := range r.Documents {
43         log.Printf("Document: %s, PageRank Score: %f", doc.DocumentName, doc.PageRankScore)
44     }
45 }
46

```

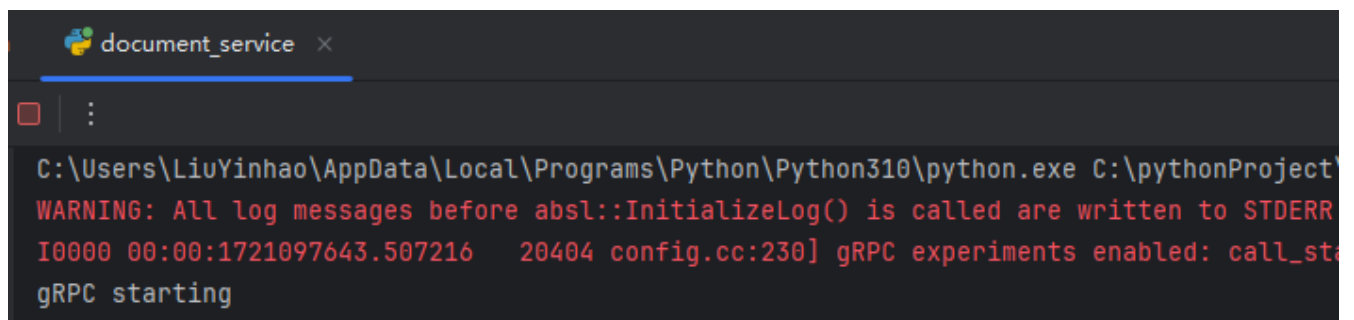
步骤 7：运行

启动 gRPC 服务器 `document_service.py`，并运行 Go 客户端，结果从 Go 客户端输出

Bash

```
1 go run document_client.go
```

python 端服务器



```

document_service x
C:\Users\LiuYinhao\AppData\Local\Programs\Python\Python310\python.exe C:\pythonProject\
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1721097643.507216 20404 config.cc:230] gRPC experiments enabled: call_sta
gRPC starting

```

Go 端客户端

```
Terminal Local x + v
PS C:\Users\LiuYinhao\GoLandProjects\gRPC> go run document_client.go
2024/07/16 10:42:36 Document: 05_技术进步对经济的推动.docx, PageRank Score: 0.647907
2024/07/16 10:42:36 Document: 20_技术在全球治理中的作用.docx, PageRank Score: 0.647907
2024/07/16 10:42:36 Document: 10_技术在交通领域的变革.docx, PageRank Score: 0.602331
2024/07/16 10:42:36 Document: 15_技术在建筑行业的应用.docx, PageRank Score: 0.601729
2024/07/16 10:42:36 Document: 04_技术在医疗中的应用.docx, PageRank Score: 0.559725
2024/07/16 10:42:36 Document: 19_技术与环境的可持续发展.docx, PageRank Score: 0.559194
2024/07/16 10:42:36 Document: 14_技术在金融服务中的应用.docx, PageRank Score: 0.383334
2024/07/16 10:42:36 Document: 09_智能技术的未来.docx, PageRank Score: 0.369831
2024/07/16 10:42:36 Document: 08_技术在农业中的应用.docx, PageRank Score: 0.339895
2024/07/16 10:42:36 Document: 13_数字技术与数据安全.docx, PageRank Score: 0.338517
2024/07/16 10:42:36 Document: 03_技术与教育.docx, PageRank Score: 0.338103
2024/07/16 10:42:36 Document: 18_未来技术的道德挑战.docx, PageRank Score: 0.323921
```

10.代码实现

环境：python3.10

源文件（Jupyter Notebook 文件）：

【金山文档 | WPS 云文档】 LDA

<https://365.kdocs.cn/l/cjmGd44UYR7J>

Python

```
1 import jieba
2 from docx import Document
3 import os
4 import numpy as np
5 from gensim import corpora
6 from gensim.models.ldamodel import LdaModel
7
8 ### 分词 ###
9 # 读取本地停用词表
10 stopwords = set(open('cn_stopwords.txt', 'r', encoding='utf-8').read
    ().splitlines()))
11
12 ### 文档读取和处理 ###
13 def read_text_from_docx(file_path):
14     """读取.docx文件并返回文本内容。"""
15     document = Document(file_path)
16     return "\n".join(para.text for para in document.paragraphs)
17
18 def preprocess_chinese_text(text):
19     """使用jieba进行中文分词，并过滤停用词。"""
20     words = jieba.cut_for_search(text)
21     return [word for word in words if word not in stopwords]
22
23 def process_documents(directory):
24     """处理目录下的所有.docx文件，并返回分词结果列表。"""
25     documents = []
26     documents_name = []
27     for filename in os.listdir(directory):
28         if filename.endswith(".docx"):
29             file_path = os.path.join(directory, filename)
30             text = read_text_from_docx(file_path)
31             segmented_text = preprocess_chinese_text(text)
32             documents.append(segmented_text)
33             documents_name.append(filename)
34     return documents, documents_name
35
36 ### LDA建模 ###
37 def prepare_corpus(documents):
```



```

38     """准备语料库和词典，用于LDA模型。"""
39     dictionary = corpora.Dictionary(documents)
40     corpus = [dictionary.doc2bow(text) for text in documents]
41     return dictionary, corpus
42
43 def lda_model(corpus, dictionary, num_topics=5, random_state=42):
44     """训练LDA模型。"""
45     return LdaModel(corpus, num_topics=num_topics, id2word=dictionary,
46                     passes=30, random_state=random_state)
47
48 ### 文档主题分布获取 ###
49 def get_topic_distribution(lda, corpus):
50     """获取每个文档的主题分布。"""
51     return [lda.get_document_topics(doc, minimum_probability=0.0) for
52             doc in corpus]
53
54 ### 文档之间的主题距离计算 ###
55 def calculate_topic_distance_abs_diff(doc_topics_i, doc_topics_j):
56     """计算两个文档之间的主题绝对距离。"""
57     topic_dist_i = dict(doc_topics_i)
58     topic_dist_j = dict(doc_topics_j)
59     all_topics = set(topic_dist_i.keys()).union(set(topic_dist_j.keys()
60     ()))
61     return sum(abs(topic_dist_i.get(topic, 0) - topic_dist_j.get(topic, 0)) for topic in all_topics)
62
63 def calculate_all_topic_distances(doc_topics):
64     """计算所有文档之间的主题距离。"""
65     num_docs = len(doc_topics)
66     distances = [[0] * num_docs for _ in range(num_docs)]
67     for i in range(num_docs):
68         for j in range(i + 1, num_docs):
69             distance = calculate_topic_distance_abs_diff(doc_topics[i], doc_topics[j])
70             distances[i][j] = distances[j][i] = distance
71     return distances
72
73 ### PageRank和点击率分析 ###
74 def convert_distance_to_similarity(distances, threshold=0.1):
75     """将距离转换为相似度，并基于相似度构建链接。"""

```

```


73     max_distance = max(max(row) for row in distances if row)
74     similarity_matrix = []
75     links = []
76     for i, row in enumerate(distances):
77         new_row = []
78         link_row = []
79         for j, dist in enumerate(row):
80             similarity = 1 - (dist / max_distance) if max_distance else
e 1
81                 new_row.append(similarity)
82                 if similarity > threshold and i != j:
83                     link_row.append(j)
84             similarity_matrix.append(new_row)
85             links.append(link_row)
86     return similarity_matrix, links
87
88 def page_rank_with_ctr(links, similarities, ctr, alpha=0.85, beta=0.7,
convergence_threshold=0.0001):
89     """结合PageRank和点击率 (CTR) 计算文档排序。"""
90     N = len(links)
91     pr = np.ones(N) / N
92     change = 1
93     while change > convergence_threshold:
94         new_pr = np.zeros(N)
95         for i in range(N):
96             link_contributions = sum(pr[j] * similarities[i][j] / len(l
inks[j]) for j in links[i] if links[j])
97             new_pr[i] = (1 - alpha) / N + alpha * (beta * link_contribu
tions + (1 - beta) * ctr[i])
98         change = np.linalg.norm(new_pr - pr)
99         pr = new_pr
100     return pr
101
102 # 主执行逻辑
103 directory = "技术1"
104 documents, document_names = process_documents(directory)
105 dictionary, corpus = prepare_corpus(documents)
106 lda = lda_model(corpus, dictionary, num_topics=5)
107 doc_topics = get_topic_distribution(lda, corpus)
108 distances = calculate_all_topic_distances(doc_topics)

```

```
109 similarities, links = convert_distance_to_similarity(distances)
110 clicks = np.array([5, 10, 15, 20, 25, 5, 10, 15, 20, 25, 5, 10, 15,
111                    20, 25, 5, 10, 15, 20, 25])
112 max_clicks = np.max(clicks)
113 ctr = clicks / max_clicks
114 pr_with_ctr = page_rank_with_ctr(links, similarities, ctr)
115 doc_pagerank = list(zip(document_names, pr_with_ctr))
116 sorted_doc_pagerank = sorted(doc_pagerank, key=lambda x: x[1], reverse
117                               =True)
118
119 # 打印排序后的结果
120 for doc_name, pr_score in sorted_doc_pagerank:
121     print(f"{doc_name}: {pr_score}")
```

11. 基于用户查询输入的实现

• 基于LDA的实现：

 仅需计算用户查询与个别文档之间的距离，而非文档集内所有可能的文档对之间的距离，大大减少了计算量。

流程：

a. 用户查询处理：

- 用户输入关键词或查询。
- 系统对查询进行预处理和主题分析，得到查询的主题分布。

b. 文档召回：

- 使用简单快速的方法召回一定数量的相关文档，例如基于关键词的索引或全文搜索。

c. 计算主题相似度：

- 对于每个召回的文档，计算它与用户查询的主题分布之间的相似度或距离。

d. 排序与展示：

- 根据主题相似度或距离对召回的文档进行排序。
- 将排序后的文档展示给用户，通常是按照与查询最相关（相似度最高或距离最近）的顺序。

e. 代码实现：

Python

```
1 import jieba
2 from docx import Document
3 import os
4 import numpy as np
5 from gensim import corpora, models
6 from gensim.models.ldamodel import LdaModel
7
8 ### 分词 ###
9 # 读取本地停用词表
10 stopwords = set(open('cn_stopwords.txt', 'r', encoding='utf-8').read()
11                  ().splitlines()))
12
13 ### 文档读取和处理 ###
14 def read_text_from_docx(file_path):
15     """读取.docx文件并返回文本内容。"""
16     document = Document(file_path)
17     return "\n".join(para.text for para in document.paragraphs)
18
19 def preprocess_chinese_text(text):
20     """使用jieba进行中文分词，并过滤停用词。"""
21     words = jieba.cut_for_search(text)
22     return [word for word in words if word not in stopwords]
23
24 def process_documents(directory):
25     """处理目录下的所有.docx文件，并返回分词结果列表。"""
26     documents = []
27     documents_name = []
28     for filename in os.listdir(directory):
29         if filename.endswith(".docx"):
30             file_path = os.path.join(directory, filename)
31             text = read_text_from_docx(file_path)
32             segmented_text = preprocess_chinese_text(text)
33             documents.append(segmented_text)
34             documents_name.append(filename)
35     return documents, documents_name
36
37 ### LDA建模 ###
```

```

38 def prepare_corpus(documents):
39     """准备语料库和词典，用于LDA模型。"""
40     dictionary = corpora.Dictionary(documents)
41     corpus = [dictionary.doc2bow(text) for text in documents]
42     return dictionary, corpus
43
44 def lda_model(corpus, dictionary, num_topics=5, random_state=42):
45     """训练LDA模型。"""
46     return LdaModel(corpus, num_topics=num_topics, id2word=dictionary,
47                     passes=30, random_state=random_state)
48
49 ### 文档主题分布获取 ###
50 def get_topic_distribution(lda, corpus):
51     """获取每个文档的主题分布。"""
52     return [lda.get_document_topics(doc, minimum_probability=0.0) for
53             doc in corpus]
54
55 ### 处理用户查询并获取查询的主题分布 ###
56 def preprocess_query(query):
57     """对用户查询进行预处理并获取主题分布。"""
58     words = jieba.cut_for_search(query)
59     query_bow = dictionary.doc2bow([word for word in words if word not
60                                     in stopwords])
61     return lda.get_document_topics(query_bow, minimum_probability=0.0)
62
63 ### 计算查询与文档之间的主题距离 ###
64 def calculate_query_document_distance(query_topics, doc_topics):
65     """计算查询与一个文档之间的主题距离。"""
66     query_dist = dict(query_topics)
67     doc_dist = dict(doc_topics)
68     all_topics = set(query_dist.keys()).union(set(doc_dist.keys()))
69     return sum(abs(query_dist.get(topic, 0) - doc_dist.get(topic, 0))
70               for topic in all_topics)
71
72 ### 归一化距离 ###
73 def normalize_distances(distances):
74     """将一系列距离归一化到0-1的区间。"""
75     max_distance = max(distances) if distances else 0
76     if max_distance == 0: # 防止除以零
77         return [0] * len(distances)

```

```

74     return [1 - (dist / max_distance) for dist in distances] # 距离越
    小, 相似度越高
75
76 def coherence(num_topics, documents):
77     lda = LdaModel(corpus, num_topics=num_topics, id2word=dictionary, p
    asses=30, random_state=42)
78     coherence_model_lda = models.CoherenceModel(model=lda, texts=docume
    nts, dictionary=dictionary, coherence='c_v')
79     coherence_lda = coherence_model_lda.get_coherence()
80     return coherence_lda
81
82 ### 结合主题相似度和点击率来计算综合得分 ###
83 def combine_scores(similarities, ctr, alpha=0.7):
84     if len(similarities) != len(ctr):
85         raise ValueError("The length of similarities and ctr must be th
    e same.")
86     return [alpha * sim + (1 - alpha) * click for sim, click in zip(si
    milarities, ctr)]
87
88 ### 主执行逻辑 ###
89 if __name__ == '__main__':
90     directory = "技术1"
91     documents, document_names = process_documents(directory)
92     dictionary, corpus = prepare_corpus(documents)
93     # x = range(10, 15)
94     # y = [coherence(i, documents) for i in x]
95     # lda = lda_model(corpus, dictionary, num_topics=max(zip(x,y),key=l
    ambda pair:pair[1]) [0])
96     # lda.save("lda_model1")
97     lda = LdaModel.load("lda_model1")
98     doc_topics = get_topic_distribution(lda, corpus)
99
100    # 假设点击数据
101    clicks = np.array([5, 10, 15, 20, 25, 5, 10, 15, 20, 25, 5, 10,
    15, 20, 25, 5, 10, 15, 20, 25])
102    max_clicks = np.max(clicks)
103    ctr = clicks / max_clicks # 归一化点击率
104
105    # 假设有用户查询
106    user_query = "技术"

```



```
107     query_topics = preprocess_query(user_query)
108
109     # 计算查询与每个文档之间的主题距离
110     doc_distances = [calculate_query_document_distance(query_topics, do
c) for doc in doc_topics]
111
112     # 归一化距离
113     normalized_distances = normalize_distances(doc_distances)
114
115     # 结合点击率和主题相似度
116     combined_scores = combine_scores(normalized_distances, ctr, alpha=
0.7)
117
118     # 排序文档基于综合得分
119     sorted_doc_distances = sorted(zip(document_names, combined_score
s), key=lambda x: x[1], reverse=True)
120
121     # 打印排序后的文档和相似度
122     for doc_name, similarity in sorted_doc_distances:
123         print(f"Document: {doc_name}, Similarity: {similarity:.4f}")
```

f. demo 输出结果

```
C:\Users\LiuYinhao\AppData\Local\Programs\Python\Python310\python.exe C:\pythonProject\LDA\LDAdemo.py
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\LIUYIN~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.492 seconds.
Prefix dict has been built successfully.
Document: 10_技术在交通领域的变革.docx, Similarity: 0.7203
Document: 05_技术进步对经济的推动.docx, Similarity: 0.7203
Document: 04_技术在医疗中的应用.docx, Similarity: 0.6580
Document: 13_数字技术与数据安全.docx, Similarity: 0.6197
Document: 15_技术在建筑行业的应用.docx, Similarity: 0.5163
Document: 11_技术在航空业的革新.docx, Similarity: 0.5011
Document: 14_技术在金融服务中的应用.docx, Similarity: 0.4559
Document: 09_智能技术的未来.docx, Similarity: 0.4142
Document: 20_技术在全球治理中的作用.docx, Similarity: 0.3647
Document: 19_技术与环境的可持续发展.docx, Similarity: 0.2723
Document: 18_未来技术的道德挑战.docx, Similarity: 0.2677
Document: 08_技术在农业中的应用.docx, Similarity: 0.2651
Document: 03_技术与教育.docx, Similarity: 0.2046
Document: 17_技术与文化的互动.docx, Similarity: 0.2011
Document: 12_技术推动的社会变化.docx, Similarity: 0.1351
Document: 02_现代技术的发展.docx, Similarity: 0.1317
Document: 07_技术与环境保护.docx, Similarity: 0.1200
Document: 16_技术对教育的长远影响.docx, Similarity: 0.1191
Document: 01_技术革命.docx, Similarity: 0.0717
Document: 06_技术和社会变革.docx, Similarity: 0.0717
```

• 基于第8部分增加用户搜索权重



在传统的应用中，PageRank 并不直接涉及内容的匹配度或用户的查询内容。然而，可以通过创新的方式将这些因素整合到一个基于 PageRank 的系统中，以增强搜索结果的相关性和个性化排序。

扩展 PageRank 以包括查询内容和匹配度

1. 查询依赖的 PageRank (Query-dependent PageRank) :

- 在这种方法中，PageRank 计算不仅基于文档间的静态链接结构，还包括查询相关的动态因素。具体来说，可以根据查询和文档内容的相关性来调整图中边的权重。
- 例如，如果用户查询与某文档高度相关，则增加该文档链接的权重。

2. 结合内容相似度和用户行为：

- 可以将文档间的相似度（基于内容或主题模型）和用户行为（如点击率）结合起来定义链接的权重。

- 这样，权重不仅反映了文档间的内容相关性，还包括了用户对文档的实际偏好，从而使得结果更符合用户的需求。

流程：

a. 预处理查询并获取其主题分布：

在 PageRank 计算前，先对用户查询进行预处理，获取查询的主题分布。

b. 计算查询与每个文档的相似度：

利用查询的主题分布与每个文档的主题分布计算相似度，这将作为动态调整链接权重的基础。

c. 调整链接权重：

根据查询与文档的相似度动态调整 PageRank 链接图中的边的权重。

d. 应用查询依赖的 PageRank：

基于调整后的权重运行 PageRank 算法，同时考虑点击率（CTR），计算每个文档的得分。

e. 代码实现：

Python

```
1 import jieba
2 from docx import Document
3 import os
4 import numpy as np
5 from gensim import corpora, models
6 from gensim.models.ldamodel import LdaModel
7
8 ### 分词 ###
9 # 读取本地停用词表
10 stopwords = set(open('cn_stopwords.txt', 'r', encoding='utf-8').read()
11                  ().splitlines()))
12
13 ### 文档读取和处理 ###
14 def read_text_from_docx(file_path):
15     """读取.docx文件并返回文本内容。"""
16     document = Document(file_path)
17     return "\n".join(para.text for para in document.paragraphs)
18
19
20 def preprocess_chinese_text(text):
21     """使用jieba进行中文分词，并过滤停用词。"""
22     words = jieba.cut_for_search(text)
23     return [word for word in words if word not in stopwords]
24
25
26 def process_documents(directory):
27     """处理目录下的所有.docx文件，并返回分词结果列表。"""
28     documents = []
29     documents_name = []
30     for filename in os.listdir(directory):
31         if filename.endswith(".docx"):
32             file_path = os.path.join(directory, filename)
33             text = read_text_from_docx(file_path)
34             segmented_text = preprocess_chinese_text(text)
35             documents.append(segmented_text)
36             documents_name.append(filename)
37     return documents, documents_name
```

```

38
39
40 ### LDA建模 ###
41 def prepare_corpus(documents):
42     """准备语料库和词典，用于LDA模型。"""
43     dictionary = corpora.Dictionary(documents)
44     corpus = [dictionary.doc2bow(text) for text in documents]
45     return dictionary, corpus
46
47
48 def lda_model(corpus, dictionary, num_topics=5, random_state=42):
49     """训练LDA模型。"""
50     return LdaModel(corpus, num_topics=num_topics, id2word=dictionary,
51                     passes=30, random_state=random_state)
52
53 ### 文档主题分布获取 ###
54 def get_topic_distribution(lda, corpus):
55     """获取每个文档的主题分布。"""
56     return [lda.get_document_topics(doc, minimum_probability=0.0) for
57             doc in corpus]
58
59 # 计算两个主题分布之间的曼哈顿距离
60 def calculate_topic_distance_abs_diff(doc_topics_i, doc_topics_j):
61     # 保证每个主题的概率被考虑到，即使某些主题在某文档中的概率为0
62     # 将主题分布转换为字典形式
63     topic_dist_i = dict(doc_topics_i)
64     topic_dist_j = dict(doc_topics_j)
65
66     # 获取所有主题的并集
67     all_topics = set(topic_dist_i.keys()).union(set(topic_dist_j.keys()
68             ()))
69
70     distance = sum(abs(topic_dist_i.get(topic, 0) - topic_dist_j.get(topic, 0))
71                     for topic in all_topics)
72     return distance
73
74 # 计算所有文档之间的主题距离
75 def calculate_all_topic_distances(doc_topics):

```

```

74     num_docs = len(doc_topics)
75     distances = [[0] * num_docs for _ in range(num_docs)]
76     for i in range(num_docs):
77         for j in range(i+1, num_docs):
78             distance = calculate_topic_distance_abs_diff(doc_topics
138 [i], doc_topics[j])
139             distances[i][j] = distances[j][i] = distance
140     return distances
141
142
143 ### 处理用户查询并获取查询的主题分布 ###
144 def preprocess_query(query, lda, dictionary):
145     query_bow = dictionary.doc2bow(jieba.cut_for_search(query))
146     query_topics = lda.get_document_topics(query_bow, minimum_probabili
147 ty=0.0)
148     return dict(query_topics)
149
150
151 # 将主题距离转换为相似度
152 # 阈值threshold用于确定两个文档之间是否存在链接，按需调整
153 def convert_distance_to_similarity(distances, threshold=0.1):
154     max_distance = max(max(row) for row in distances if row)
155     similarity_matrix = []
156     links = []
157     for i, row in enumerate(distances):
158         new_row = []
159         link_row = []
160         for j, dist in enumerate(row):
161             similarity = 1 - (dist / max_distance) if max_distance else
162 e 1
163             new_row.append(similarity)
164             if similarity > threshold and i != j:
165                 link_row.append(j)
166         similarity_matrix.append(new_row)
167         links.append(link_row)
168     return similarity_matrix, links
169
170
171 def calculate_query_similarity(query_topics, doc_topics):
172     num_docs = len(doc_topics)

```

```

111     distances = [[0] for _ in range(num_docs)]
112     for i in range(num_docs):
113         distance = calculate_topic_distance_abs_diff(query_topics, doc_
            topics[i])
114         distances[i] = distance
115     max_distance = max(distances) if distances else 0
116     query_similarity = [1 - (dist / max_distance) if max_distance els
        e 1 for dist in distances]
117     return query_similarity
118
119
120 def adjust_link_weights(doc_topics, query_similarity, links, base_weigh
    t=0.01):
121     adjusted_weights = []
122     for i, topics in enumerate(doc_topics):
123         doc_similarity = query_similarity[i]
124         # 将基础权重添加到与查询相关的文档上
125         row_weights = [base_weight + doc_similarity if j in links[i] el
            se 0 for j in range(len(doc_topics))]
126         adjusted_weights.append(row_weights)
127     return adjusted_weights
128
129
130 def coherence(num_topics, documents):
131     lda = LdaModel(corpus, num_topics=num_topics, id2word=dictionary, p
        asses=30, random_state=42)
132     coherence_model_lda = models.CoherenceModel(model=lda, texts=docume
        nts, dictionary=dictionary, coherence='c_v')
133     coherence_lda = coherence_model_lda.get_coherence()
134     return coherence_lda
135
136
137 # 计算PageRank
138 def page_rank_with_ctr(links, similarities, adjusted_weights, ctr, alph
    a=0.85, beta = 0.95, convergence_threshold=0.0001):
139     N = len(links)
140     pr = np.ones(N) / N # 初始化PR值, 总和为1
141     change = 1
142     w1 = 0.6 # adjusted_weights的系数
143     w2 = 0.4 # similarities的系数

```



```

144     # 进行迭代直到收敛
145     while change > convergence_threshold:
146         new_pr = np.zeros(N)
147         for i in range(N):
148             link_contributions = 0
149             for j in links[i]: # 遍历节点i的所有出链节点j
150                 if len(links[j]) > 0: # 避免除以零
151                     # 结合 adjusted_weights 和 similarities 作为权重
152                     link_weight = w1 * adjusted_weights[j][i] + w2 * si
153                     link_contributions += pr[j] * link_weight / len(links[j])
154             new_pr[i] = (1 - alpha) / N + alpha * (beta*link_contributions + (1-beta)*ctr[i])
155             # 归一化新的PageRank值，确保它们的总和为1
156             new_pr /= np.sum(new_pr) # 归一化步骤
157             change = np.linalg.norm(new_pr - pr)
158             pr = new_pr
159     return pr
160
161
162
163 ### 主执行逻辑 ###
164 if __name__ == '__main__':
165     directory = "技术1"
166     documents, document_names = process_documents(directory)
167     dictionary, corpus = prepare_corpus(documents)
168
169     # 第一次运行需要训练LDA模型
170     # x = range(5, 15)
171     # y = [coherence(i, documents) for i in x]
172     # lda = lda_model(corpus, dictionary, num_topics=max(zip(x,y),key=lambda pair:pair[1])[0])
173     # lda.save("lda_model1")
174
175     # 之后可以直接加载已经训练好的模型
176     lda = LdaModel.load("lda_model1")
177     doc_topics = get_topic_distribution(lda, corpus)
178
179     # 假设点击数据

```

```

180     clicks = np.array([5, 10, 15, 20, 25, 5, 10, 15, 20, 25, 5, 10,
181                        15, 20, 25, 5, 10, 15, 20, 25])
182     max_clicks = np.max(clicks)
183     ctr = clicks / max_clicks # 归一化点击率
184
185     # 假设有用户查询，获取用户查询主题分布
186     query = "技术"
187     query_topics = preprocess_query(query, lda, dictionary)
188
189     query_similarity = calculate_query_similarity(query_topics, doc_top
190     ics)
191
192     # 计算查询与每个文档之间的主题距离
193     distances = calculate_all_topic_distances(doc_topics)
194
195     similarities, links = convert_distance_to_similarity(distances)
196
197     adjusted_weights = adjust_link_weights(doc_topics, query_similarit
198     y, links)
199
200     pr = page_rank_with_ctr(links, similarities, adjusted_weights, ctr)
201
202     # 按PageRank值对文档进行排序
203     pagerank_score = pr
204     doc_pagerank = list(zip(document_names, pagerank_score))
205
206     # 按PageRank分数降序排序
207     sorted_doc_pagerank = sorted(doc_pagerank, key=lambda x: x[1], rev
208     erse=True)
209
210     # 打印排序后的结果
211     for doc_name, pr_score in sorted_doc_pagerank:
212         print(f"{doc_name}: {pr_score}")

```

f. 关键点解释：

- 链接权重计算：

$$w_1 * adjusted_weights[j][i] + w_2 * similarities[j][i]$$

- `adjusted_weights[j][i]`：用户的搜索与文档之间的相关性

- `similarities[j][i]`: 文档之间的相似度
- w_1, w_2 : 权重系数

使用”加和“结合两个权重而非使用乘积：避免一个权重极低时对整体影响过大的问题。

- 归一化计算：

$$\text{new_pr} /= \text{np.sum}(\text{new_pr})$$

- 通过将所有新计算的 PageRank 值除以它们的总和，这一步骤确保了新的 PageRank 数组的总和为 1。
- 确保了无论网络的链接结构如何变化，所有页面的 PageRank 值总和始终为 1，符合随机游走模型，确保了算法的输出可以直接作为页面或文档的“重要性”或“排名”的概率解释。

g. demo 输出结果

```
C:\Users\LiuYinhao\AppData\Local\Programs\Python\Python310\python.exe C:\pythonProject\PageRank\LDA_Pagerank_query.py
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\LIUYIN~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.418 seconds.
Prefix dict has been built successfully.
Document: 14_技术在金融服务中的应用.docx, PageRank Score: 0.06800
Document: 05_技术进步对经济的推动.docx, PageRank Score: 0.06701
Document: 10_技术在交通领域的变革.docx, PageRank Score: 0.06457
Document: 15_技术在建筑行业的应用.docx, PageRank Score: 0.06355
Document: 20_技术在全球治理中的作用.docx, PageRank Score: 0.06148
Document: 09_智能技术的未来.docx, PageRank Score: 0.06095
Document: 04_技术在医疗中的应用.docx, PageRank Score: 0.05765
Document: 07_技术与环境保护.docx, PageRank Score: 0.05452
Document: 13_数字技术与数据安全.docx, PageRank Score: 0.05159
Document: 03_技术与教育.docx, PageRank Score: 0.05069
Document: 19_技术与环境的可持续发展.docx, PageRank Score: 0.04936
Document: 11_技术在航空业的革新.docx, PageRank Score: 0.04772
Document: 01_技术革命.docx, PageRank Score: 0.04764
Document: 08_技术在农业中的应用.docx, PageRank Score: 0.04307
Document: 17_技术与文化的互动.docx, PageRank Score: 0.04256
Document: 18_未来技术的道德挑战.docx, PageRank Score: 0.04115
Document: 12_技术推动的社会变化.docx, PageRank Score: 0.03657
Document: 02_现代技术的发展.docx, PageRank Score: 0.03401
Document: 06_技术和社会变革.docx, PageRank Score: 0.03076
Document: 16_技术对教育的长远影响.docx, PageRank Score: 0.02717
```

12. 训练广泛的 LDA 模型



构建一个足够通用的模型，在面对用户输入关键词的多样性和不可预测性时能够捕捉文档集合中的主题分布是有必要的，然后利用这个模型来帮助排序查询结果。

数据来源：

THUCNews 中文文本分类数据集的处理，该数据集包含 84 万篇新闻文档，总计 14 类；在数据集的基础上可以进行文本分类、词向量的训练等任务。数据集的下载地址：<http://thuc.c.thunlp.org/>

数据训练：

停用词表： stopwords

Python

```
1 # 读取本地停用词表
2 stopwords = set(line.strip() for line in open('stopwords.txt', 'r', encoding='utf-8').read().splitlines())
3
4 # 读取文本数据
5 def read_texts(directory):
6     for filename in os.listdir(directory):
7         if filename.endswith('.txt'):
8             file_path = os.path.join(directory, filename)
9             with open(file_path, encoding='utf-8') as file:
10                 yield file.read()
11
12 # 使用jieba进行分词和词性标注
13 def preprocess_chinese_text(text):
14     words = jieba.cut_for_search(text, HMM=True)
15     # 过滤停用词和进行词性选择
16     filtered_words = [word for word in words if word not in stopwords
17                       and word.strip() and not word.isspace()]
18     return filtered_words
19
20 # 训练LDA模型
21 def lda_model(corpus, dictionary, num_topics=5):
22     # 设置训练LDA模型的参数
23     lda = models.ldamulticore.LdaMulticore(corpus, num_topics=num_topics,
24                                             id2word=dictionary, passes=30, random_state=42, workers=4)
25     return lda
26
27 directory = "C:/pythonProject/train_model/THUCNews/THUCNews/cn_train_data"
28
29 processed_texts = list(read_texts(directory))
30
31 # 对文本进行分词
32 segmented_texts = [preprocess_chinese_text(text) for text in processed_texts]
33
34 # 创建字典和语料库
35 dictionary = corpora.Dictionary(segmented_texts)
```

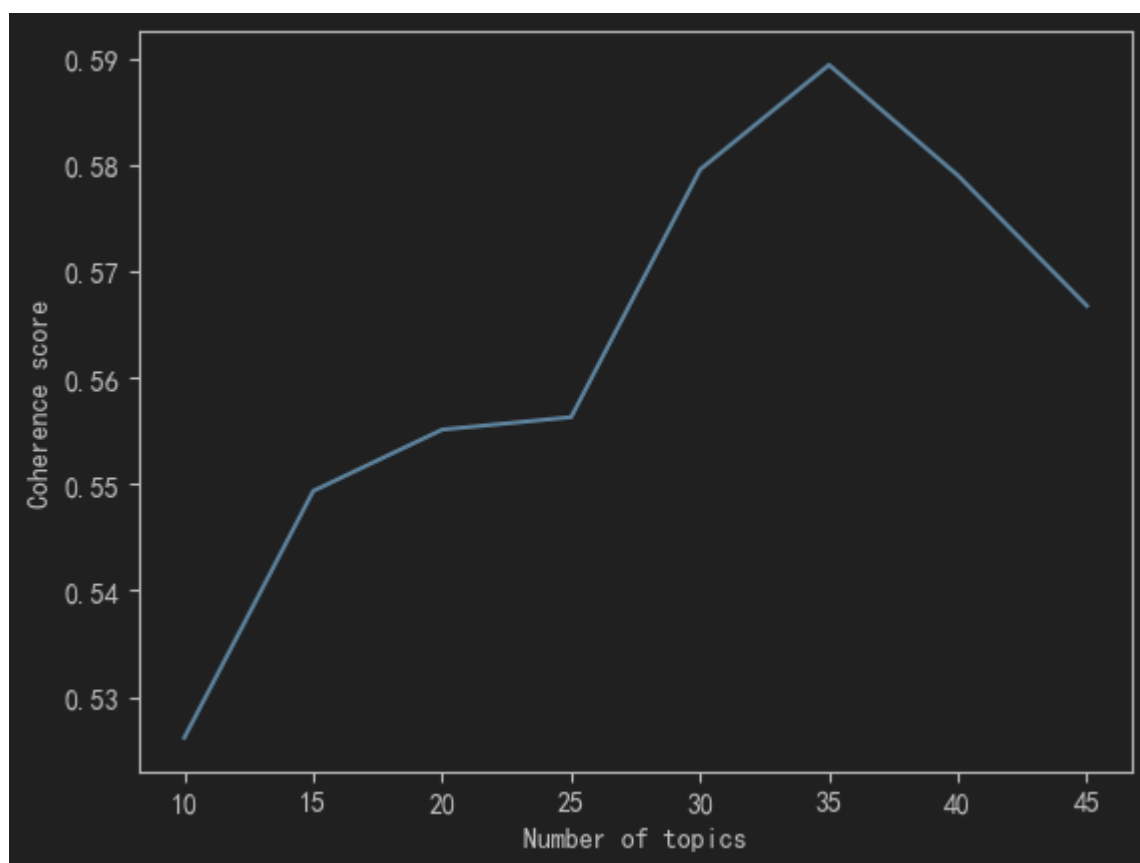
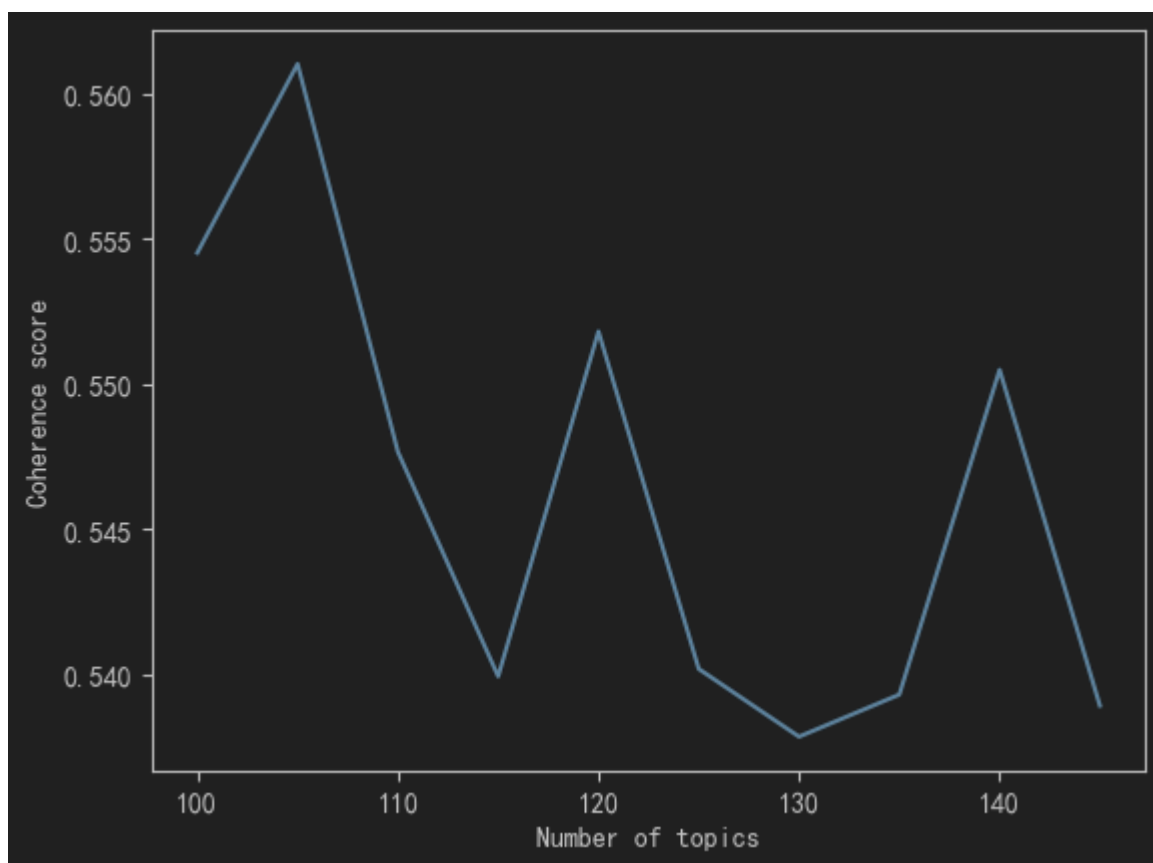
```
34 corpus = [dictionary.doc2bow(text) for text in segmented_texts]
35 lda = models.ldamulticore.LdaMulticore(corpus, num_topics=105, id2word=
    dictionary, passes=30, random_state=42, workers=4)
36
37 #保存模型
38 lda.save('lda_model_105')
```

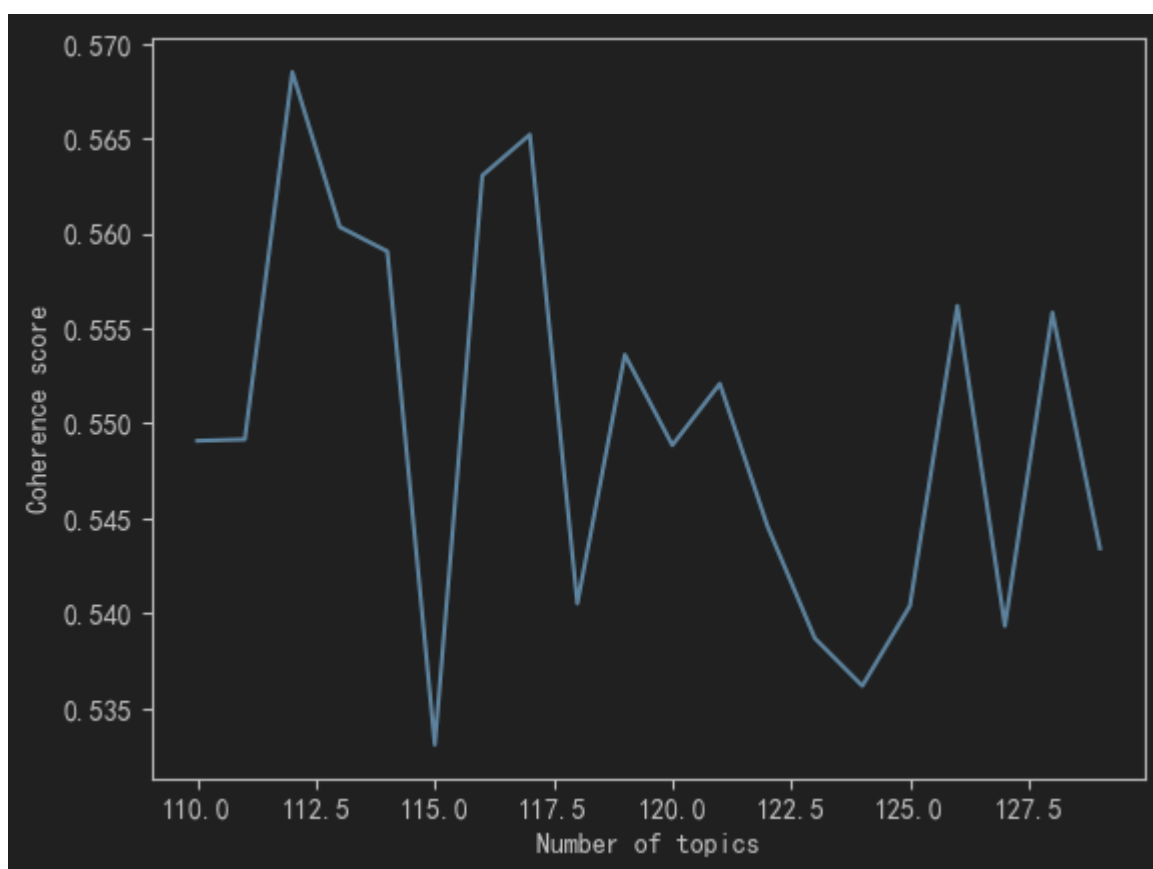
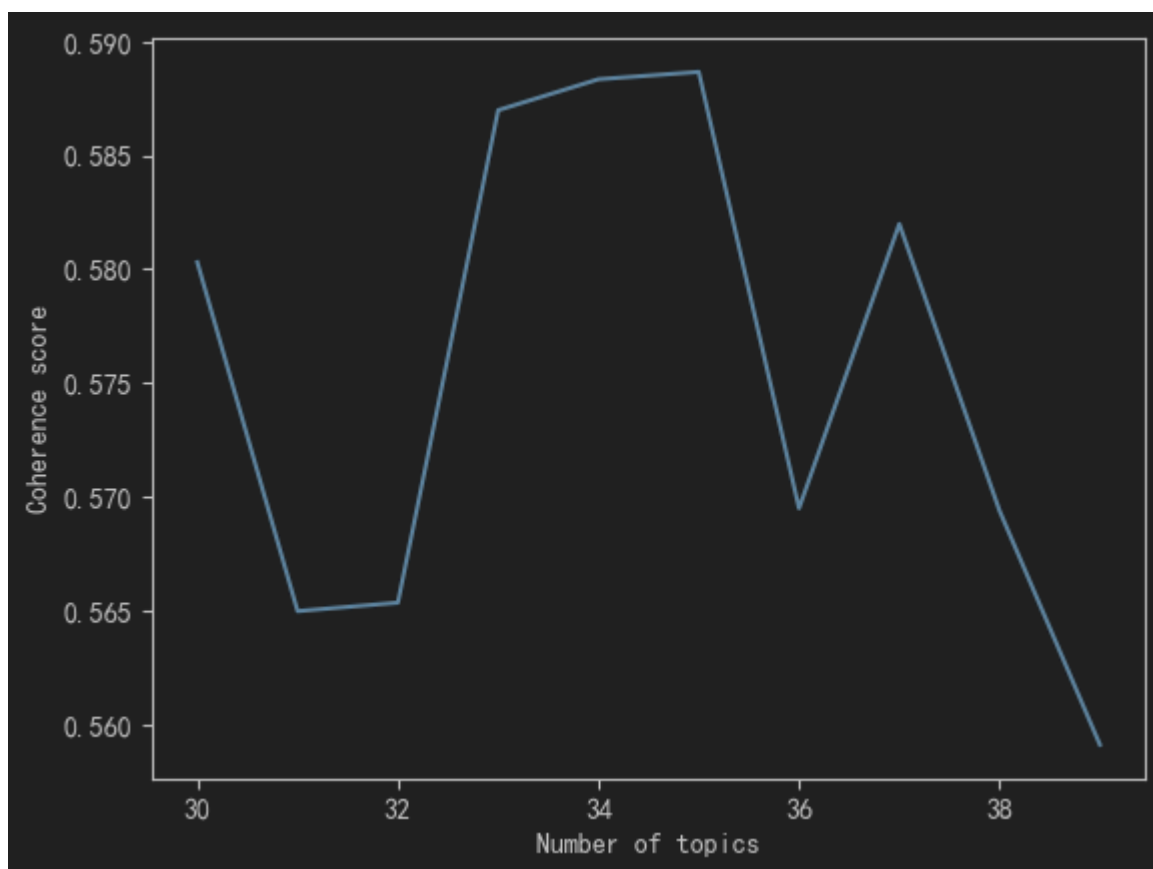
模型评价：

主题连贯性分数(Coherence Score)是一种客观的衡量标准，它基于语言学的分布假设：具有相似含义的词往往出现在相似的上下文中。如果所有或大部分单词都密切相关，则主题被认为是连贯的。

Python

```
1 def coherence(num_topics, corpus, dictionary, segmented_texts):
2     lda = models.ldamulticore.LdaMulticore(corpus, num_topics=num_topics,
3         id2word=dictionary, passes=30, random_state=42, workers=4)
4     coherence_model_lda = models.CoherenceModel(model=lda, texts=segmented_texts,
5         dictionary=dictionary, coherence='c_v')
6     coherence_lda = coherence_model_lda.get_coherence()
7     return coherence_lda
8
9 x = range(100, 150, 5)
10 y = [coherence(i, corpus, dictionary, segmented_texts) for i in x]
11 plt.plot(x, y)
12 plt.xlabel('Number of topics')
13 plt.ylabel('Coherence score')
14 plt.rcParams['font.sans-serif'] = ['SimHei']
15 matplotlib.rcParams['axes.unicode_minus'] = False
16 plt.show()
17 # 打印最佳num_topic
18 max(zip(x, y), key=lambda pair: pair[1])[0]
19 # 106
```





13.在线学习 (Online Learning)

在线学习也称为增量学习或适应性学习，是指对一定顺序下接收数据，每接收一个数据，模型会对它进行预测并对当前模型进行更新，然后处理下一个数据。

• 使用 Gensim 实现在线更新的 LDA 模型

LDA 实现支持通过连续的文档流进行模型更新。

Python

```
1 from gensim.models import LdaModel
2 from gensim.corpora import Dictionary
3
4 # 创建一个词典和语料库
5 dictionary = Dictionary()
6 lda_model = LdaModel(corpus=None, id2word=dictionary, num_topics=10, passes=1)
7
8 # 在新文档到来时更新模型
9 for new_documents in stream_of_documents():
10     # 更新字典
11     dictionary.add_documents(new_documents)
12     # 创建新的语料库
13     new_corpus = [dictionary.doc2bow(doc) for doc in new_documents]
14     # 更新 LDA 模型
15     lda_model.update(new_corpus)
```

引用

1. [Brin, S.; Page, L. \(1998\). "The anatomy of a large-scale hypertextual Web search engine" \(PDF\). *Computer Networks and ISDN Systems*. **30** \(1 – 7\): 107 – 117. *CiteSeerX* 10.1.1.115.5930. doi:10.1016/S0169-7552\(98\)00110-X. ISSN 0169-7552. S2CID 7587743. Archived \(PDF\) from the original on 2015-09-27.](#)

2. Massimo Franceschet (2010). "PageRank: Standing on the shoulders of giants". *arXiv:1002.2858 [cs.IR]*.
3. Kim M, Kim D. A Suggestion on the LDA-Based Topic Modeling Technique Based on ElasticSearch for Indexing Academic Research Results. *Applied Sciences*. 2022; 12(6):3118. <https://doi.org/10.3390/app12063118>
4. [🔗 LDA主题模型简介及Python实现-CSDN博客](#)
5. [🔗 LDA主题模型评价指标汇总_Ida模型困惑度的值一般为多少-CSDN博客](#)
6. [🔗 通俗理解LDA主题模型\(2014年版\)-CSDN博客](#)
7. [🔗 A Suggestion on the LDA-Based Topic Modeling Technique Based on ElasticSearch for Indexing Academic Research Results](#)
8. <http://pasa-bigdata.nju.edu.cn/achievement/KeyExtract.html>
9. [🔗 用python对单一微博文档进行分词——jieba分词（加保留词和停用词）_python分词处理和停用词表设置-CSDN博客](#)
10. <https://dev.to/sishaarrao/demystifying-the-pagerank-algorithm>
11. [🔗 gRPC和Protobuf扩展 - Go语言高级编程](#)
12. [🔗 python golang中grpc 使用示例代码详解-腾讯云开发者社区-腾讯云](#)
13. <http://thuctc.thunlp.org/#%E8%8E%B7%E5%8F%96%E9%93%BE%E6%8E%A5>
14. [🔗 GitHub - duoergun0729/nlp: 兜哥出品 <一本开源的NLP入门书籍>](#)
15. [🔗 【主题建模】基于连贯性分数（Coherence Score）的主题建模评估-CSDN博客](#)