Link Prediction 实验

211250109 赵政杰

2023 年 10 月 10 日

复现论文：Link Prediction for Wikipedia Articles as a Natural Language Inference Task

1. 论文摘要 abstract 和 introduction 翻译

abstract 译文

链接预测任务（link prediction task）对于自动理解大型知识库的结构至关重要。在本文中，我们在 2023 年数据科学和高级分析竞赛（DSAA-2023 Competition）“高效且有效的链接预测”任务中展示了解决此问题的系统，该系统包含 948,233 个训练数据和 238,265 个用于公共测试的语料库。本文介绍了一种在维基百科文章中进行链接预测的方法，将其表述为自然语言推理（NLI）任务。受自然语言处理和理解领域最新进展的启发，我们将链接预测作为一项 NLI 任务，其中两篇文章之间存在链接被视为前提，任务是根据文章中的信息确定该前提是否成立。我们基于维基百科文章的链接预测任务的句对分类来实现我们的系统。我们的系统在公共和私人测试集上分别获得了 0.99996 Macro F1-score 和 1.00000 Macro F1-score。我们 UIT-NLP 团队在私人测试集上的表现排名第三，与第一名和第二名的成绩持平。我们的代码是公开的，用于研究目的。

introduction 译文

维基百科（Wikipedia）是世界上最大的协作式百科全书，已成为获取各种主题知识的宝贵资源。维基百科拥有数百万篇涵盖不同主题的文章，提供了一个不断扩展的庞大信息库。然而，尽管维基百科的规模令人印象深刻，但维基百科的文章之间的相互链接并不总是全面的，导致信息连通性方面存在差距。

链接预测是网络分析中的一项基本任务，旨在根据现有连接预测给定网络中缺失的链接。在维基百科的背景下，链接预测变得尤为重要，因为它可以帮助增强百科全书的导航性、提高信息可访问性并促进对相关主题的更深入理解。DSAA-2023 挑战赛重点关注应用于维基百科文章的链接预测任务。此外，DSAA-2023 挑战赛的重点是提出类网络数据结构中的链路预测方法，例如网络重建和网络开发，使用维基百科文章作为主要数据源。

自然语言推理的任务是在给定“前提”的情况下确定“假设”是真（蕴含）、假（矛盾）还是不确定（中性）。维基百科文章任务的链接预测被定义为给出维基百科网络的稀疏子图并预测两个维基百科页面之间是否存在链接。在本文中，我们利用自然语言推理任务和维基百科文章的链接预测任务之间固有的相似性，将 NLI 中常用的句对分类 (SPC) 技术应用于链接预测任务的特定上下文。

本文通过结合基于 NLI 的 SPC 和预处理技术来解决维基百科文章的链接预测挑战。维基百科中链接预测的传统方法通常依赖于基于图的算法或文本相似性度量，这可能会忽略文章文本中嵌入的微妙关系。通过集成句对分类和预处理技术，我们旨在更有效地捕获维基百科文章中的语义和上下文信息，以提高链接预测准确性。

我们的贡献总结如下：

1. 首先，我们采用高效的数据预处理技术来清洗从维基百科获得的评论。这些技术的利用有助于提高数据的整体质量，并在为链接预测任务训练模型之前改进相关信息的提取。

2. 其次，我们通过将 NLI 与维基百科文章的链接预测之间的相似性结合起来，采用了自然语言推理中广泛使用的 SPC 技术来进行链接预测。
3. 最后，我们在这项任务上取得了最好的成绩，在公开测试中得分为 0.99996，在私人测试中得分为 1.00000，分别排名第八和第三。

2. 问题描述

维基百科文章链接预测任务是指基于给定的维基百科数据集，预测一对文章节点 (u, v) 之间是否存在边。更准确地说，给定维基百科文章网络的稀疏子图，确定维基百科上下文中两个页面 u 和 v 之间是否存在链接。

3. 输入、输出、模型算法描述

输入

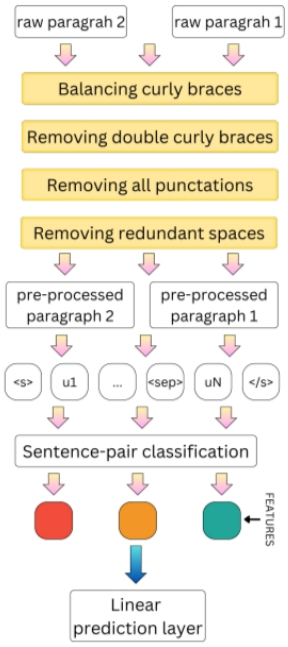
1. train.csv 文件：它包含节点对以及它们之间是否存在边的 label。该文件有四列：id、id1、id2 和 label。id 列表示配对标识符，id1 和 id2 是文章节点标识符，label 表示两节点之间是否存在边（0 或 1）。
2. nodes.zip 文件：此文件是 nodes.tsv 的压缩版本，包含每个文章节点的信息。它由两列组成：id（文章节点标识符）和 text （节点的文本描述）。

输出

输出模型的预测结果，输出格式参照 sample_submission.csv，应当包含两类，一列是 pair ID，即配对标识符；一列为 label，表示两节点是否存在边（0 或 1）。

模型算法描述

方法架构图



预处理算法

该数据集由维基百科对象的主要内容和维基百科内的 CSS 代码组成。然而，这些 CSS 代码被认为是噪声，不需要进行训练。因此，去除它们至关重要。为了实现这一目标，我们采用两种算法，即平衡花括号算法和删除双花括号算法。

1. **平衡花括号算法**（Balancing Curly Braces）

   算法步骤：

   1. 分别统计左花括号和右花括号的数量，并将结果存储在 `opening_count` 和 `closing_count` 变量中。
   2. 平衡花括号：
      - 如果 `opening_count` 大于 `closing_count`，说明需要删除一些左花括号。
        - 进入一个 `while` 循环，条件是 `opening_count` 大于 `closing_count`。
        - 删除找到的第一个左花括号，将 `opening_count` 减 1。
        - 重复此过程，直到左右花括号相等。
      - 如果 `closing_count` 大于 `opening_count`，说明需要删除一些右花括号。
        - 进入一个 `while` 循环，条件是 `closing_count` 大于 `opening_count`。
        - 删除找到的第一个右花括号，将 `closing_count` 减 1。
        - 重复此过程，直到左右花括号相等。
   3. 返回平衡后的文本。

```
procedure BALANCECURLYBRACES(text)
        opening_count ← text.count('{')
        closing_count ← text.count('}')
        if opening_count > closing_count then
                while opening_count > closing_count do
                        index ← text.find('{')
                        if index  -1 then
                                text = text[:index] + text[index + 1:]
                                opening_count ← opening_count - 1
        else if closing_count > opening_count then
                while closing_count > opening_count do
                        index ← text.rfind('}')
                        if index  -1 then
                                text = text[:index] + text[index + 1:]
                                closing_count ← closing_count - 1
        return text.strip()
```

2. **删除双花括号算法**（Remove Double Curly Braces）

   算法步骤：

   1. 创建一个空栈 `stack` 和一个空字符串 `clean_text`。

2. 遍历文本字符
   - 如果是左花括号，则进行压栈。
   - 如果是右花括号
     - 如果栈非空且栈顶为左花括号，则将其弹栈。
   - 如果是其他字符，如果栈为空，则将其添加到 `clean_text` 中。
3. 返回去除双花括号后的 `clean_text`。

```
procedure REMOVEDOUBLECURLYBRACES(text)
        stack ← []
        clean_text ← ''
        for char in text do
                if char = '{' then
                        stack.push(char)
                else if char = '}' then
                        if not isEmpty(stack) and
                                top(stack) = '{' then
                                stack.pop(stack)
                else
                        if isEmpty(stack) then
                                clean_text ← clean_text + char
        return clean_text
```

3. **利用正则表达式库删除所有标点符号以及冗余空格**

模型算法

利用自然语言推理任务和维基百科文章的链接预测任务之间固有的相似性，实现了基于 XLM-Roberta 的句对分类模型来进行链接预测。

模型算法步骤：

1. **分词（Tokenization）**
   - XLM-Roberta 首先接受多语言文本作为输入，并使用分词器将文本分割成离散的 tokens（标记）。Tokens 可以是单词、子词（如词根或词缀）、标点符号等，取决于分词器和分词规则。
2. **建立 Transformer 模型**
   - XLM-Roberta 构建在 Transformer 架构之上。Transformer 是一种深度学习模型，特别适用于处理序列数据。
   - 模型包括多个 Transformer 层，每个层都包括多头自注意力机制和前馈神经网络。
   - 这些 Transformer 层通过堆叠来构建深度神经网络，用于学习文本的表示。
3. **预处理（Pretraining）**
   - 在预训练阶段，XLM-Roberta 使用大规模的多语言文本数据进行训练，执行通常是建立“掩盖语言模型”的任务。

- 在掩盖语言模型中，模型被要求预测每个 token 中一小部分被随机遮盖的部分，以从上下文中理解其含义。

4. **微调（Fine-tuning）**
   - 预训练完成后，XLM-Roberta 可以通过微调来适应特定的下游 NLP 任务，如文本分类、命名实体识别、文本翻译等。
   - 微调通常包括在模型的顶部添加适当的输出层，以匹配任务的类别数或标签，然后进行有监督的训练。

     在此任务中，我们为模型整合了一个线性预测层来计算最终的输出。
   - 在微调过程中，模型的参数会针对具体任务的数据集进行调整，以适应任务的需求。
5. **应用**
   - 将经过微调的 XLM-Roberta 模型用于链接预测任务，来判断两篇文章之间是否具有关联。

4. 评价指标及其计算公式

使用**宏平均 F1 分数**（Macro F1-score）进行结果的评估。

计算公式如下：

$$\text{Macro F1-score} = \frac{1}{N}\sum_{i=1}^{N}\frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

其中：

- N 是类别（链接类型）的总数。在此任务中为 2。
- Precision_i 是第 i 个类别的精确度（Precision），定义为模型正确预测为该类别的文章对数目除以总的预测为该类别的文章对数目。
- Recall_i 是第 i 个类别的召回率（Recall），定义为模型正确预测为该类别的文章对数目除以总的真实属于该类别的文章对数目。

上述公式计算了每个类别的 F1 分数，然后对所有类别的 F1 分数取平均值，得到宏平均 F1 分数。这个指标用于综合评估模型在多类别分类任务中的性能，而不受类别不平衡的影响。

5. 对比方法及这些对比方法的引用论文出处

本文通过结合基于 NLI 的 SPC 和预处理技术来解决维基百科文章的链接预测挑战。

对比传统方法，它们通常依赖于基于图的算法或文本相似性度量，然而这可能会忽略文章文本中嵌入的微妙关系，导致预测准确率不高。

对比方法出处：

1. Link prediction based on local random walk
2. Link prediction based on graph neural networks
3. Link Prediction in Heterogeneous Social Networks

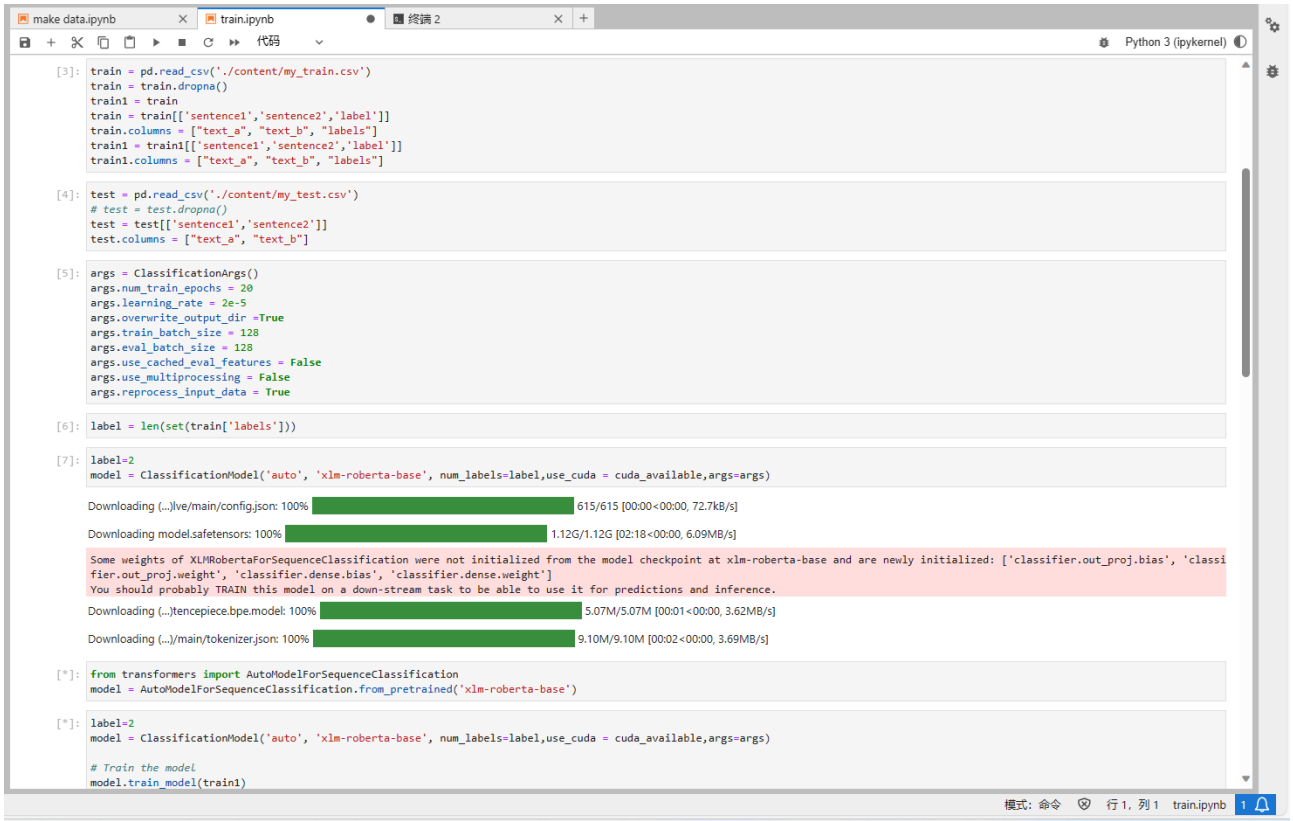
同时，文章还对比其本身有预处理技术和无预处理技术的分数，结果如下：

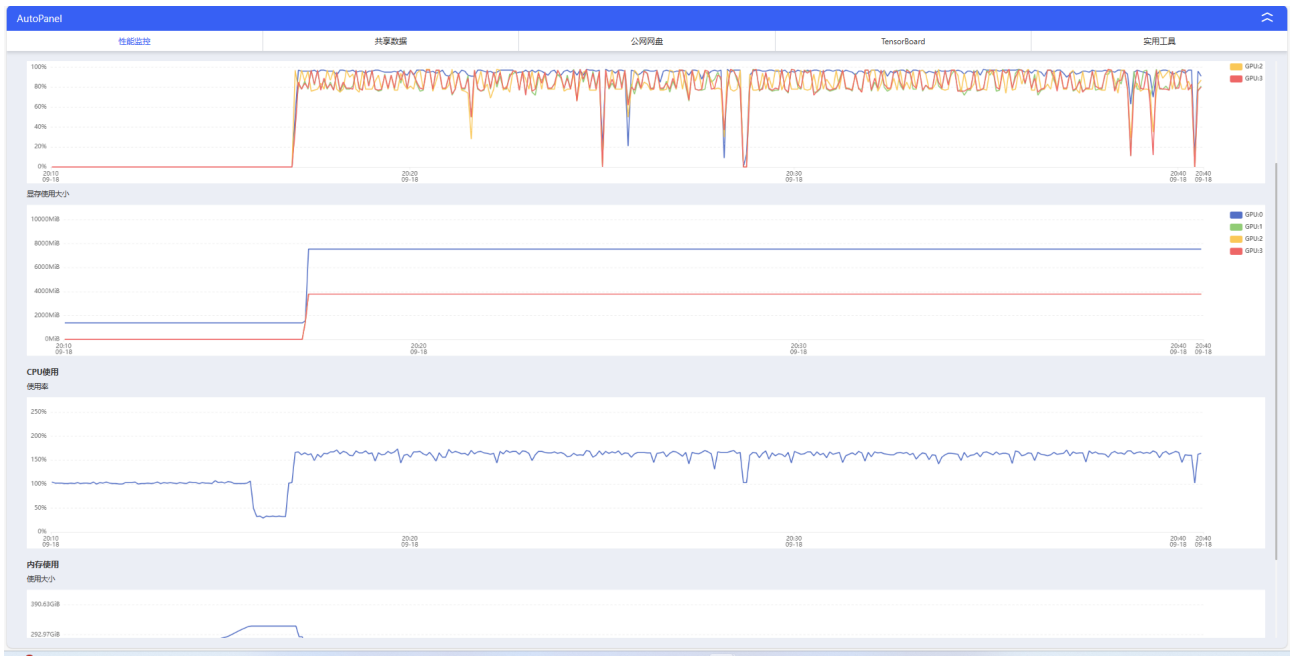
	Public test	Private test
With Preprocessing techniques		
Our approach	**0.99996**	**1.00000**
Without Preprocessing techniques		
Our approach	0.97680	0.97663

6. 结果

我们选择在 AutoDL 平台上进行实验复现。

我们租用了 4 张 3080ti 的卡进行训练，训练过程如图：





最终模型评估的结果如下：

```
result, model_outputs, wrong_predictions = model.eval_model(train[10000:10000+150])
output = [np.argmax(i,axis = 0) for i in model_outputs ]
print('accuracy',accuracy_score(train[10000:10000+150]['labels'],output))
print('weighted',f1_score(train[10000:10000+150]['labels'],output,average = 'weighted'))
print('macro',f1_score(train[10000:10000+150]['labels'],output,average = 'macro'))
```

```
  0%|          | 0/150 [00:00<?, ?it/s]
Running Evaluation:   0%|          | 0/15 [00:00<?, ?it/s]
accuracy 0.9533333333333334
weighted 0.9530529834229189
macro 0.952572383576494
```

得到的结果与论文的结果基本相近。碍于算力，训练的模型并未到达文中的最佳效果。