NLP

笔记本:	我的第一个笔记本		
创建时间:	2023/10/7 16:30	**更新时间:**	2023/10/11 10:07
作者:	3446994995@qq.com		
URL:	https://chat.openai.com/c/fd9d33cb-df76-42b7-9603-8f3c5616b2b1		

NLP

前置知识学习

multi-label classification&multi-class classification

Multi-Label Classification（多标签分类），多标签例如：一个生活场景图片，上面有果酱、有面包、有人、有猫、有桌子等等，那么这个照片就可以是一个多标签，对于这类数据的归类称为多标签分类。

Multi-Class Classification（多类分类），就是每个数据只有一个类别（label）例如，一个单纯的猫的图片，一张单纯的狗的图片，对这类数据进行分类。

Hierarchical multi-label text classification (HMTC)任务是什么?

就是先预测最高级别的父标签，然后在逐级向下预测更具体的子标签，以此对文本进行分类。
举个例子:

- **新闻主题层次结构**
  - 政治

    - 国内政治
    - 国际政治
- 经济
    - 财经市场
    - 公司新闻
- 文化
    - 电影
    - 音乐
- 科技
    - 科学
    - 互联网

先预测类似与政治、经济这样的父类标签，在进行逐步向下的预测子标签，例如，先预测了政治这一个父标签，然后在预测是国内政治还国际政治这个子标签，最后完成预测。

如何让计算机感知到并有能力处理人类的语言？人类语言中的每个词都需要输入给计算机处理分析吗？

用NLP（自然语言处理）技术，

- 和计算机视觉一样，要给予一定数量的处理好的数据
- 然后建立关键词表（这个也可以说明不需要输入人类语言中的每个词）
- 然后训练模型以及训练好模型后的应用

虽然不需要每个词都要输入，但是所选的关键词要具有代表性，所用的文本数据也要具有一定的代表性，不可随便选择。
例如：不能将“吃”作为“我要进焦糖工作室”这句话的关键词！！！！！

NLP流程

1. **文本数据采集：** 任何机器学习都需要大量的数据，以此来进行学习与训练！！！总之，万机器学习之源都是从找数据开始的！！！
2. **文本预处理：** 比如说：“我喜欢吃吃吃吃吃吃吃吃！”中“吃这个字重复的次数太多”，可以进行简单的合并、再例如“我喜欢吃小区门口左边的第三条街的往里面走三十米的那家有着韭菜肉馅的红色包子铺的包子”这个文本我们自己看起来都难，更不用说是电脑了（电脑也就算数比我快那么一点点，其他的，嘻嘻），所以，我们要进行分词处理（很重要的预处理）方便模型的训练。当然还有停用词去除和词干化等多个步骤。

3. **建立词汇表：** 计算机不需要分析每个词汇单元，但需要建立一个词汇表，其中包含了任务相关的单词和词汇。这有助于限制处理的词汇范围，并提高计算效率。
4. **模型训练：** 利用预处理的文本数据，然后就可以用深度学习来进行训练咯，根据数据的种类以及码主设计的输出来确定最后的模型是什么样的功能。
5. **模型应用**：
    1. **语言理解：** 训练好的NLP模型能够理解文本中的语言结构和含义。它们可以识别词汇、句法、语法、语义和上下文信息，使计算机能够理解人类语言。
    2. **语言生成：** 计算机不仅可以理解语言，还可以生成自然语言文本。这在聊天机器人、自动生成文档和答案等应用中很有用。
    3. **处理大规模数据：** 计算机可以处理大规模的文本数据，从中提取信息、分析趋势、生成摘要等。

分词&去除停用词是干什么的?

分词

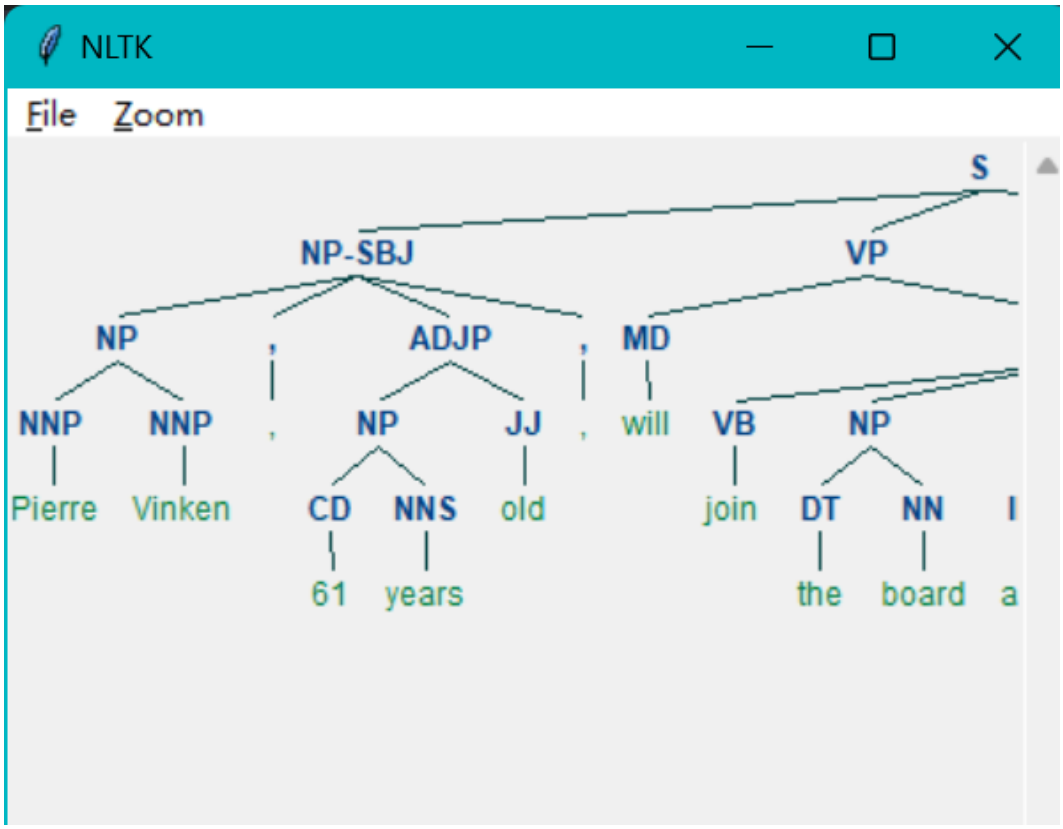
- 分词是将连续的文本分割成单词、短语或符号的过程
- 例如，“我喜欢学习，想要进焦糖”分词为“我” “喜欢” “学习” “想要” “进” “焦糖” 。
- **好可惜，刚刚用nltk分词了一下我写的那句话，发现是怎么进怎么出，也试了试简单的汉语，发现都是这样的，看到官网的例子，发现英语可以，所以，嘿嘿嘿**

```
sentence = "I like to eat the buns of the red bun shop " \
           "with leek meat filling 30 meters down the third street " \
           "on the left side of the entrance of the community"
# sentence = "At eight o'clock on Thursday morning ,Arthur didn't feel very good."
tokens = nltk.word_tokenize(sentence)
print(tokens)
```

```
['I', 'like', 'to', 'eat', 'the', 'buns', 'of', 'the', 'red', 'bun', 'shop', 'with', 'leek', 'meat', 'filling', '30', 'meters', 'down', 'the', 'third', 'street', 'on', 'the', 'left', 'side', 'of',
 'the', 'entrance', 'of', 'the', 'community']
```

好好玩，此外，我也尝试了nltk的其他功能，如词性标注、命名实体识别、展示Treebank里面的第一个句子的解析树

```
词性标注 [('At', 'IN'), ('eight', 'CD'), ("o'clock", 'NN'), ('on', 'IN'), ('Thursday',
 'NNP'), ('morning', 'NN')]
命名实体识别 (S
  At/IN
  eight/CD
  o'clock/NN
  on/IN
  Thursday/NNP
```



使用nltk时遇到的问题：

- 在使用nltk之前要输入以下代码来下载文件，

```
nltk.download('punkt')  # 下载标点符号的数据# nltk.download('stopwords')  # 下载停用词的数据nltk.download('wordnet')  # 下载WordNet词典的数据（用于词义处理） nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')# nltk.download('treebank')
```

- 原来的“001”已经不行了，通过下面的代码我查了查文件名称

```
t = treebank.parsed_sents('wsj_001.mrg')[0]
```

```
file_ids = treebank.fileids()
print(file_ids)
#print(nltk.data.path)
```

发现要改成“0001”

去除停用词

- 停用词指的是，在文本中经常出现但本身不携带信息的词汇，
  如：“的”“是”“在”“和”等。
- 去除停用词是将这些常见的无意义词汇从文本中删除的过程，以减少文本的噪声和冗余信息。
- 停用词的去除有助于提高文本处理任务的效率，同时可以提高文本分析的准确性，因为它们通常不包含有关文本内容的重要信息。

```
分词 ['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', ',', 'Arthur', 'did', "n't",
 'feel', 'very', 'good', '.']
停用词： ['eight', "o'clock", 'Thursday', 'morning', ',', 'Arthur', "n't", 'feel', 'good',
 '.']
```

词向量/word2vec是什么？语料库（corpus）是什么?

词向量（Word Vectors）和Word2Vec是自然语言处理（NLP）领域中的关键概念，而语料库（Corpus）是与NLP研究和任务密切相关的一个重要资源。

1. **词向量（Word Vectors）**：
   - 词向量是一种将单词映射到实数向量的表示方法。每个单词都被表示为多维向量，这些向量捕捉了单词之间的语义关系。
   - 词向量的目标是将自然语言中的词汇转化为计算机可以理解的数值形式，以便在NLP任务中使用。词向量通常基于分布式假设，即在文本中相似上下文中出现的词汇在向量空间中会更加接近。
2. **Word2Vec**:
   是一种常用的词向量模型，它使用神经网络方法来学习单词的连续向量表示，包括两个模型：连续词袋（Continuous Bag of Words，CBOW）和跳字模型（Skip-gram）。
   在这里我尝试用代码的输出看例子，用的是CBOW模型，但是在现在学习阶段看不太懂，我先附上截图 （遗留问题，有待解决！！！！）

```
good的向量表示: [-8.2426788e-03  9.2993546e-03 -1.9766092e-04 -1.9672776e-03
  4.6036290e-03 -4.0953159e-03  2.7431131e-03  6.9399667e-03
  6.0654259e-03 -7.5107957e-03  9.3823504e-03  4.6718074e-03
  3.9661191e-03 -6.2435055e-03  8.4599778e-03 -2.1501661e-03
  8.8251876e-03 -5.3620026e-03 -8.1294207e-03  6.8245577e-03
  1.6711927e-03 -2.1985101e-03  9.5135998e-03  9.4938539e-03
 -9.7740479e-03  2.5052286e-03  6.1566923e-03  3.8724565e-03
  2.0227861e-03  4.3050051e-04  6.7363022e-04 -3.8206363e-03
 -7.1402504e-03 -2.0888734e-03  3.9238976e-03  8.8186832e-03
  9.2591504e-03 -5.9759379e-03 -9.4026709e-03  9.7643761e-03
  3.4297847e-03  5.1661157e-03  6.2823440e-03 -2.8042626e-03
  7.3227026e-03  2.8302716e-03  2.8710032e-03 -2.3803711e-03
 -3.1282497e-03 -2.3701428e-03  4.2764354e-03  7.6057913e-05
 -9.5842788e-03 -9.6655441e-03 -6.1481954e-03 -1.2856961e-04
  1.9974159e-03  9.4319675e-03  5.5843499e-03 -4.2906976e-03
  2.7831554e-04  4.9643586e-03  7.6983096e-03 -1.1442233e-03
  4.3234206e-03 -5.8143805e-03 -8.0419064e-04  8.1000496e-03
 -2.3600650e-03 -9.6634552e-03  5.7792594e-03 -3.9298222e-03
 -1.2228728e-03  9.9805165e-03 -2.2563506e-03 -4.7570658e-03
 -5.3293873e-03  6.9808890e-03 -5.7088733e-03  2.1136617e-03
 -5.2556610e-03  6.1207130e-03  4.3573068e-03  2.6063537e-03
 -1.4910841e-03 -2.7460647e-03  8.9929365e-03  5.2157734e-03
 -2.1625208e-03 -9.4703101e-03 -7.4260519e-03 -1.0637427e-03
 -7.9494715e-04 -2.5629092e-03  9.6827196e-03 -4.5852186e-04
  5.8737611e-03 -7.4475883e-03 -2.5060750e-03 -5.5498648e-03]
与good最相似的单词:
so: 0.13149002194404602
You: 0.0679759606719017
right.: 0.04157736524939537
job.: -0.01351496484130621
good.: -0.04461707919836044
```

```python
# 导入必要的库
from gensim.models import Word2Vec
import logging

# 配置日志以查看训练过程中的进展
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

# 准备训练数据（示例数据）
sentences = [
    "You are right.".split(),
    "You are so good.".split(),
    "good job.".split()
]

# 初始化Word2Vec模型并进行训练
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, sg=0)
# 获取单词的向量表示
word = 'good'
if word in model.wv:
    vector = model.wv[word]
    print(f"{word}的向量表示: {vector}")
else:
    print(f"{word}不在词汇表中")

# 查找与给定单词最相似的单词
similar_words = model.wv.most_similar(word, topn=5)
print(f"与{word}最相似的单词:")
for similar_word, score in similar_words:
    print(f"{similar_word}: {score}")
```

3. **语料库（Corpus）（就像torchvision中的内置数据库差不多吧）**：
    - 语料库是指包含大量文本文档的集合或数据库。这些文档可以包括书籍、文章、新闻、网页、社交媒体帖子等等，涵盖了多种主题和领域。
    - 语料库是用于训练NLP模型、进行文本分析和研究自然语言的重要数据源。研究人员和工程师可以使用语料库来构建词汇表、训练词向量、开发文本分类器、进行情感分析等任务。
    - 大规模的语料库对于许多NLP任务至关重要，因为它们能够更好地捕捉词汇和语法的多样性，从而提高了模型的性能。

总之，词向量是一种用于将单词转化为数值表示的方法，有助于计算机理解和处理自然语言。Word2Vec是一种流行的词向量模型。而语料库是包含大量文本数据的集合，用于训练NLP模型和进行文本分析。这些概念在NLP中起到了关键作用，帮助计算机处理和理解人类语言。

Attention机制

注意力机制（Attention Mechanism）是一种在深度学习和自然语言处理领域广泛应用的重要技术，它模拟了人类处理信息时的注意力过程。注意力机制允许模型根据输入数据的不同部分分配不同的注意力权重，以便更好地处理信息。

以下是注意力机制的关键概念和应用：

1. **注意力权重（Attention Weights）**：

- 在注意力机制中，针对每个输入元素（如序列中的单词或图像中的像素）都会计算一个注意力权重。这个权重表示了在模型执行特定任务时，模型应该关注输入中的哪些部分。
- 注意力权重通常是概率分布，其和为1，这意味着模型为每个输入元素分配了一个权重，以指示其在输出中的贡献。
  - **举例 图像处理中的注意力**：
  - 在计算机视觉中，注意力机制也被广泛用于处理图像数据。图像中的注意力机制可以用于选择图像中的特定区域或特征，以执行诸如图像标注、目标检测和图像生成等任务。

2. **自注意力（Self-Attention）**：

   - 自注意力机制是一种常见的注意力机制，用于处理序列数据，如自然语言中的句子或文本。它允许模型在处理一个序列时，根据序列中的其他元素分配不同的注意力权重。
   - 通过自注意力，模型可以理解哪些部分的输入与当前处理的部分相关，从而更好地捕捉上下文信息。
   - **举例 Transformer模型**：
     - Transformer是一种深度学习模型架构，广泛使用自注意力机制来处理序列数据，特别是在机器翻译和文本生成等NLP任务中表现出色。
     - 在Transformer中，自注意力层被用于编码输入序列和生成输出序列，其注意力权重可以根据输入序列的不同部分动态调整。

总之，注意力机制是深度学习中的一个重要工具，允许模型动态地关注输入中的不同部分，以更好地处理信息。它在NLP和计算机视觉等领域中被广泛用于提高模型的性能和效果。

recurrent network

Recurrent Neural Network（循环神经网络，RNN）是一种深度学习模型，用于处理序列数据。与传统神经网络不同，RNN 具有一种循环结构，使它能够在处理序列数据时保持状态或记忆。

RNN 的主要特点包括：

1. **循环结构：** RNN 中的循环结构使得信息可以在序列中传递，因此 RNN 能够捕捉到序列数据中的时序依赖关系。每个时间步都接收一个输入并生成一个输出，同时维护一个内部隐藏状态（hidden state），该状态会随着时间步的前进而更新。

2. **权重共享：** RNN 在每个时间步都使用相同的权重参数，因此它在不同时间步之间共享权重。这使得 RNN 具有一种递归的特性，可以用于处理可变长度的序列数据。

3. **用途广泛：** RNN 可以应用于各种序列数据的任务，包括自然语言处理（NLP）、语音识别、时间序列分析、图像生成等。它们在序列建模和预测中非常有用。

然而，传统的 RNN 具有一些问题，如难以捕捉长距离的依赖关系和梯度消失问题（在训练中，梯度逐渐变小，使得远处时间步的信息难以传播到当前时间步）。为了克服这些问

题，出现了一些改进型的 RNN 架构，包括长短时记忆网络（LSTM）和门控循环单元（GRU），它们更好地捕捉长距离依赖并解决了梯度消失问题。

RNN 和其变种在很多序列建模任务中表现出色，但也有一些限制。在处理长序列时，仍然存在梯度消失和梯度爆炸等问题。为了克服这些问题，出现了其他架构，如 Transformer，它在自然语言处理领域取得了重大突破。尽管如此，RNN 仍然是理解和处理序列数据的基本概念，对于某些特定任务仍然非常有用。

LSTM&Bi-LSTM

LSTM（Long Short-Term Memory）和Bi-LSTM（Bidirectional Long Short-Term Memory）都是深度学习中用于处理序列数据的重要模型，特别适用于自然语言处理（NLP）和时间序列任务。

以下是 LSTM 和 Bi-LSTM 的解释和比较：

LSTM（Long Short-Term Memory）：

- LSTM 是一种循环神经网络（RNN）的变种，旨在解决传统 RNN 中的梯度消失和梯度爆炸问题，同时能够处理长序列数据。
- LSTM 引入了三个门控单元：遗忘门（forget gate）、输入门（input gate）、输出门（output gate）。这些门控单元允许 LSTM 控制信息流，并决定在当前时间步保留哪些信息、忽略哪些信息。
- LSTM 的内部状态（隐藏状态）可以捕获长期依赖关系，使其适用于需要理解上下文信息的任务，如机器翻译和文本生成。

Bi-LSTM（Bidirectional Long Short-Term Memory）：

- Bi-LSTM 是 LSTM 的扩展，它同时考虑了序列数据的过去和未来信息。在每个时间步，Bi-LSTM 同时有一个前向（forward）LSTM 和一个后向（backward）LSTM，它们分别从序列的起始和末尾开始向中间移动。
- 这意味着Bi-LSTM 在当前时间步可以获取来自过去和未来的上下文信息，有助于更好地理解整个序列。它在序列标注、情感分析等任务中表现出色，因为它可以捕获到单词或标记的双向上下文信息。

比较：

- LSTM 和 Bi-LSTM 都可以处理序列数据，但 Bi-LSTM 具有更强的上下文建模能力，因为它同时考虑了过去和未来信息。
- Bi-LSTM 在某些任务中表现出色，特别是在需要全局上下文信息的情况下。然而，由于其复杂性，它通常需要更多的计算资源。
- LSTM 在许多序列任务中仍然很有用，尤其是在资源有限或需要更快训练和推理速度的情况下。

总之，LSTM 和 Bi-LSTM 是处理序列数据的强大工具，选择使用哪个取决于任务需求和计算资源。Bi-LSTM 在许多NLP任务中表现出色，但对于某些任务，LSTM 仍然是一个有效的选择。