

期末项目

中山大学计算机学院

人工智能

本科生实验报告

(2022学年春季学期)

课程名称：Artificial Intelligence

教学班级	冯班	专业 (方向)	计算机科学与技术 (系统结构)
学号	20337188	姓名	邓理华

一、概述

反事实陈述描述了一个未发生或不可能发生的事件，以及若该事件发生可能会有什么结果。反事实推断是人类的高级思维模式之一，人类可以根据知识及因果关系推断出反事实事件可能的原因。AI如何进行反事实推断，是因果推断研究的重要议题。本次实验通过训练NB, KNN, LR, 双向LSTM等模型，识别句子是否为反事实陈述，是后续进行反事实分析的基础。

二、实验原理

NB

贝叶斯分类算法是统计学中的一种概率分类方法，朴素贝叶斯分类是贝叶斯分类中最简单的一种。其分类原理就是利用贝叶斯公式根据某特征的先验概率计算出其后验概率，然后选择具有最大后验概率作为该特征所属的类。之所以称之为朴素，是因为贝叶斯分类只做最原始、最简单的假设：所有的特征之间是相对独立的。

KNN

所谓 K 近邻算法，即是给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的K个实例，这K个实例的多数属于某个类，就把该输入实例分类到这个类中。近邻是根据两个数据点之间的距离度量来定义的。最近邻分类器通常基于测试样本与指定训练样本之间的欧氏距离。

LR

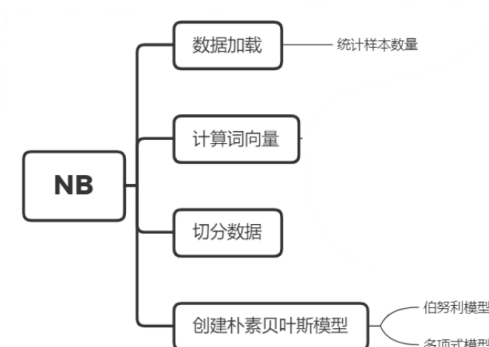
逻辑回归模型是一个二分类的对数线性模型，是经典的二分类算法。面对一个回归或者分类问题，建立代价函数，然后通过优化方法迭代求解出最优的模型参数，然后测试验证我们这个求解的模型的好坏。Logistic 回归虽然名字里带“回归”，但是它实际上是一种分类方法，主要用于两分类问题（即输出只有两种，分别代表两个类别）

RNN-双向LSTM

人类是基于经验或者已有的知识开始对新事物的学习 —— 正如你不会一开始就研究深度学习、计算机视觉与自然语言处理这些课程，而往往需要先掌握 Python 编程、数据分析库的经验。但是传统的神经网络并不能做到这点，看起来也像是一种巨大的弊端。例如，假设你希望对电影中的每个时间点的时间类型进行分类。传统的神经网络应该很难来处理这个问题 —— 使用电影中先前的事件推断后续的事件。RNN 解决了这个问题。RNN 是包含循环的网络，允许信息的持久化。

LSTM 是一种 RNN 特殊的类型，在很多问题，LSTM 都取得相当巨大的成功，并得到了广泛的使用。LSTM 通过刻意的设计来避免长程依赖问题。记住长程的信息成为了 LSTM 的默认行为。

三、方法



数据加载

由于使用的是Google Colab环境，需要连接上Google硬盘

计算词向量

one-hot

使用一个V维向量表示一篇文章，向量的长度V为词汇表的大小。1表示存在对应的单词，0表示不存在。

TF-IDF

TF指的是某一个给定的词语在该文件中出现的频率。这个数字是对词数的归一化，以防止它偏向长的文件。同一个词语在长文件里可能会比短文件有更高的词数，而不管该词语重要与否。

TF-IDF数值为IDF与TF的乘积，把IDF作为权重。某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

切分数据

调用train_test_split对数据进行切分，其中测试集占0.2，即训练集与测试集比例为 8:2

创建朴素贝叶斯模型

伯努利模型

伯努利朴素贝叶斯就是先验概率为伯努利分布的朴素贝叶斯。假设特征的先验概率为二元伯努利分布。

$$P(X_j = x_{jl} | Y = C_k) = \frac{x_{jl} + \lambda}{m_k + 2\lambda}$$

在伯努利模型中，每个特征的取值只有 `True` 和 `False`。在文本分类中，就是一个特征有没有出现在一个文档中。

多项式模型

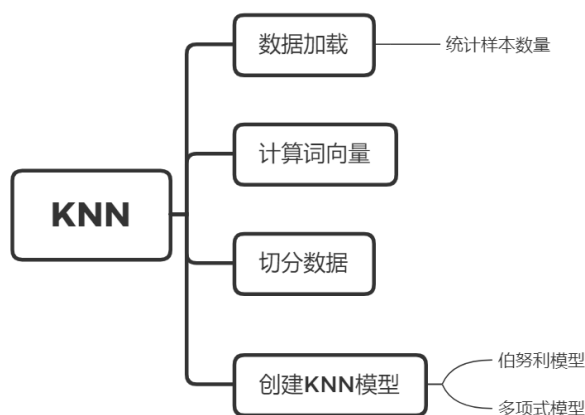
多项式朴素贝叶斯就是先验概率为多项式分布的朴素贝叶斯。假设特征是由一个简单多项式分布生成的。多项式分布可以描述各种类型样本出现次数的概率，因此多项式朴素贝叶斯非常适合用于描述出现次数或者出现次数比例的特征。该模型常用于文本分类，特征值表示的是次数。公式如下：

$$P(X_j = x_{jl} | Y = C_k) = \frac{x_{jl} + \lambda}{m_k + n\lambda}$$

其中：

- $P(X_j = x_{jl} | Y = C_k)$ 是第 k 个类别的第 j 维特征的第 l 个取值的条件概率
- m_k 是训练集中输出为第 k 类的样本个数
- n 为数据的维度, λ 是一个大于 0 的常数, 当 $\lambda = 1$ 时, 为拉普拉斯平滑。

调库实现：使用词向量和编码标签进行朴素贝叶斯模型建立并进行预测得到概率。我们使用 scikit-learn 库创建基本模型。然后，利用 fit() 拟合函数对模型进行训练。然后，可以使用训练好的模型进行预测。



此处略去与上述模型相似的流程。

创建KNN模型

在 KNN 中，通过计算对象间距离，来作为各个对象之间的非相似性指标，避免了对对象之间的匹配问题，在这里距离一般使用：欧氏距离或曼哈顿距离：

$$\text{欧氏距离: } d(x, y) = \sqrt{\sum_{k=1}^n (x_k - y_k)^2}$$

$$\text{曼哈顿距离: } d(x, y) = \sqrt{\sum_{k=1}^n |x_k - y_k|}$$

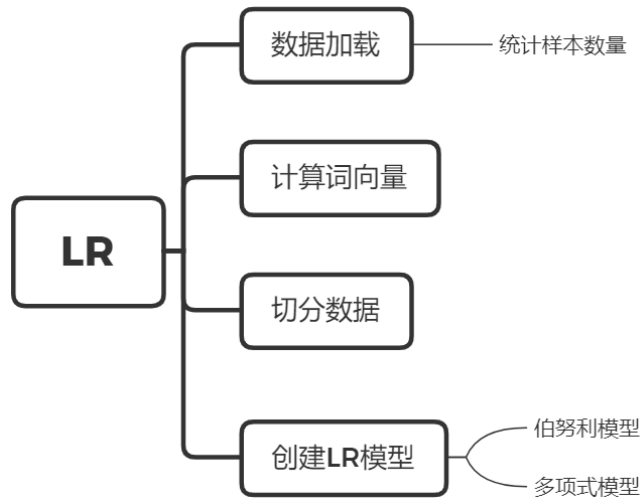
同时，KNN 通过依据 K 个对象中占优的类别进行决策，而不是单一的对象类别决策。这两点就是 KNN 算法的优势。

算法具体描述为：

1. 计算测试数据与各个训练数据之间的距离
2. 按照距离的递增关系进行排序
3. 选取距离最小的 K 个点
4. 确定前 K 个点所在类别的出现频率
5. 返回前 K 个点中出现频率最高的类别作为测试数据的预测分类

有时，其他测量方法可能更适合特定的环境，包括：曼哈顿、切比雪夫和汉明距离。

调库实现：使用词向量和编码标签进行 K 近邻模型建立。我们使用 scikit-learn 的 KNeighborsClassifier 类创建基本模型，并将 K 的值传递给模型。然后，利用 fit() 拟合函数对模型进行训练。然后，可以使用训练好的模型进行预测。

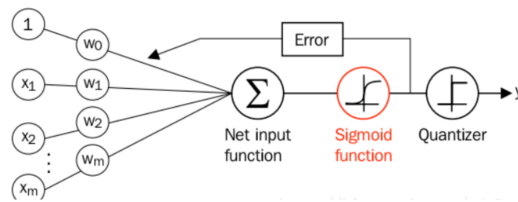


此处略去与上述模型相似的流程。

创建LR模型

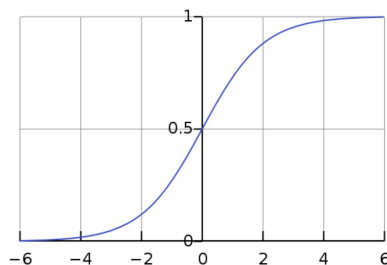
逻辑回归中最核心的概念是Sigmoid函数，可以看成逻辑回归的激活函数。

下图是逻辑回归网络：



对数几率函数 (Sigmoid)

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$



通过对数几率函数的作用，我们可以将输出的值限制在区间[0, 1]上， $p(x)$ 则可以用来表示概率 $p(y=1|x)$ ，即当一个 x 发生时， y 被分到1那一组的概率。可是，等等，我们上面说 y 只有两种取值，但是这里却出现了一个区间[0, 1]，为什么呢？其实在真实情况下，我们最终得到的 y 的值是在 [0, 1] 这个区间上的一个数，然后我们可以选择一个阈值，通常是 0.5，当 $y > 0.5$ 时，就将这个 x 归到 1 这一类，如果 $y < 0.5$ 就将 x 归到 0 这一类。但是阈值是可以调整的，比如说一个比较保守的人，可能将阈值设为 0.9，也

就是说有超过90%的把握，才相信这个x属于 1这一类。了解一个算法，最好的办法就是自己从头实现一次。下面是逻辑回归的具体实现。

常规步骤

寻找h函数（即预测函数）

构造J函数（损失函数）

想办法（迭代）使得J函数最小并求得回归参数（ θ ）

函数h(x)的值有特殊的含义，它表示结果取1的概率，于是可以看成类1的后验估计。因此对于输入x分类结果为类别1和类别0的概率分别为：

$$P(y = 1 \mid x; \theta) = h\theta(x)$$

$$P(y = 0 \mid x; \theta) = 1 - h\theta(x)$$

代价函数

逻辑回归一般使用交叉熵作为代价函数。

交叉熵是对“出乎意料”的度量。神经元的目标是去计算函数 y, 且 $y = y(x)$ 。但是我们让它取而代之计算函数 a, 且 $a = a(x)$ 。假设我们把 a 当作 y 等于 1 的概率，1-a 是 y 等于 0 的概率。那么，交叉熵衡量的是我们在知道 y 的真实值时的平均“出乎意料”程度。当输出是我们期望的值，我们的“出乎意料”程度比较低；当输出不是我们期望的，我们的“出乎意料”程度就比较高。

交叉熵代价函数如下所示：

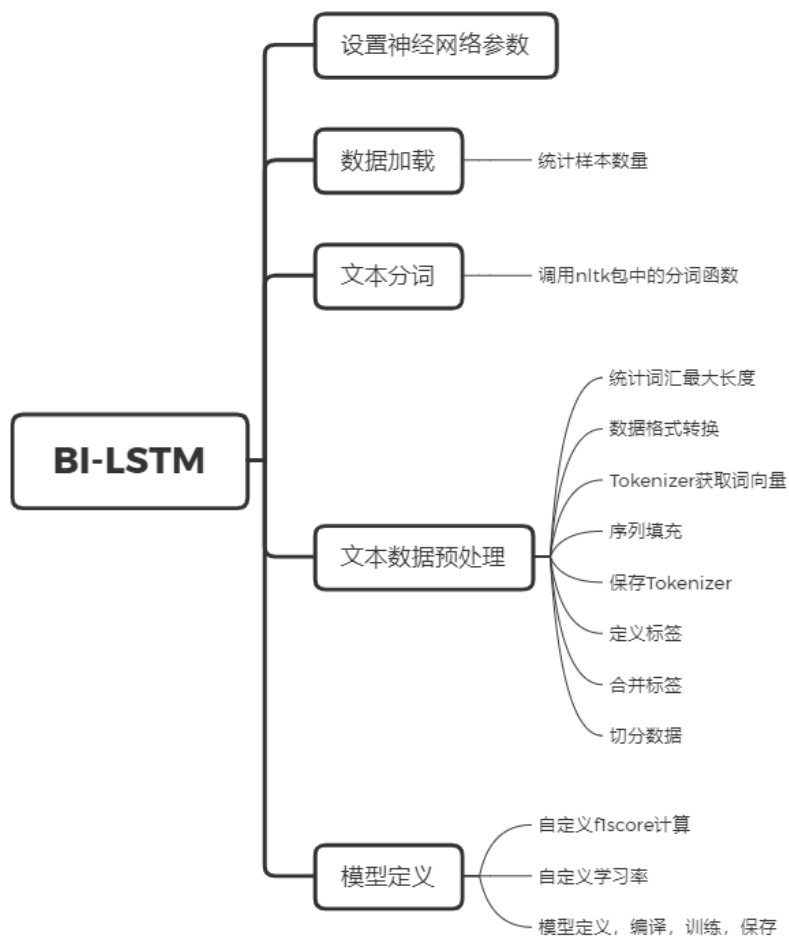
$$J(w) = -l(w) = - \sum_{i=1}^n y^{(i)} \ln(\phi(z^{(i)})) + (1 - y^{(i)}) \ln(1 - \phi(z^{(i)}))$$

$$J(\phi(z), y; w) = -y \ln(\phi(z)) - (1 - y) \ln(1 - \phi(z))$$

调库实现：使词向量和编码标签进行逻辑回归模型建立并进行预测得到概率。我们使用 scikit-learn 的 LogisticRegression类创建基本模型。然后，利用 fit() 拟合函数对模型进行训练。然后，可以使用训练好的模型进行预测。



RNN-双向LSTM



设置神经网络参数

为了方便统一设置，我们在这里指定神经网络训练时所需要的参数，在后面的代码中可以直接引用。

数据加载（同上）

文本分词

调用nltk包中的分词函数。

NLTK，全称Natural Language Toolkit，自然语言处理工具包，是NLP研究领域常用的一个Python库，由宾夕法尼亚大学的Steven Bird和Edward Loper在Python的基础上开发的一个模块，至今已有超过十万行的代码。这是一个开源项目，包含数据集、Python模块、教程等。

文本数据预处理

按以下步骤进行数据预处理：

1. 统计词汇最大长度
2. 数据格式转换为字符串格式
3. Tokenizer获取词向量
 - a. 实例化Tokenizer，设置字典中最大词汇数为30000

b. Tokenizer会自动过滤掉一些符号比如：!"#\$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n

4. 序列填充

- 把序列设定为 `max_length` 的长度，超过 `max_length` 的部分舍弃，不到 `max_length` 则补 `0`
- `padding='pre'` 在句子前面进行填充，`padding='post'` 在句子后面进行填充，本实验选择句前填充

5. 保存Tokenizer为 json 文件

6. 定义标签

7. 合并标签

8. 切分数据

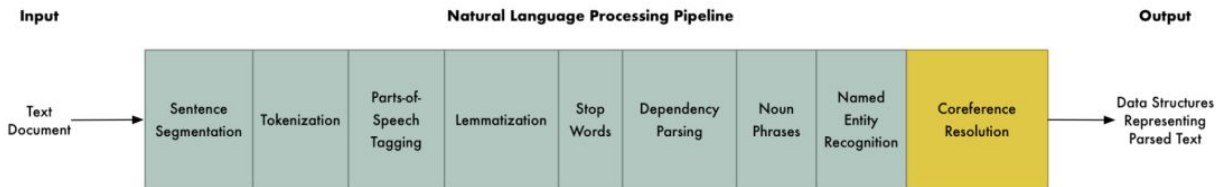
- 调用 `train_test_split` 对数据进行切分，其中测试集占 `0.2`，即训练集与测试集比例为 8:2

由于时间原因未能实现的数据处理方法

ELMo

• pipeline

- 在机器学习中做任何复杂的事情通常意味着需要建立一条流水线 (pipeline)。这个想法是把你的问题分解成非常小的部分，然后用机器学习来分别解决每个部分，最后通过把几个互相馈送结果的机器学习模型连接起来，这样你就可以解决非常复杂的问题。
- 这正是我们要运用在 NLP 上的策略。我们将把理解英语的过程分解成小块，然后看看每个小块是如何工作的。



• NLP库

◦ spaCy库

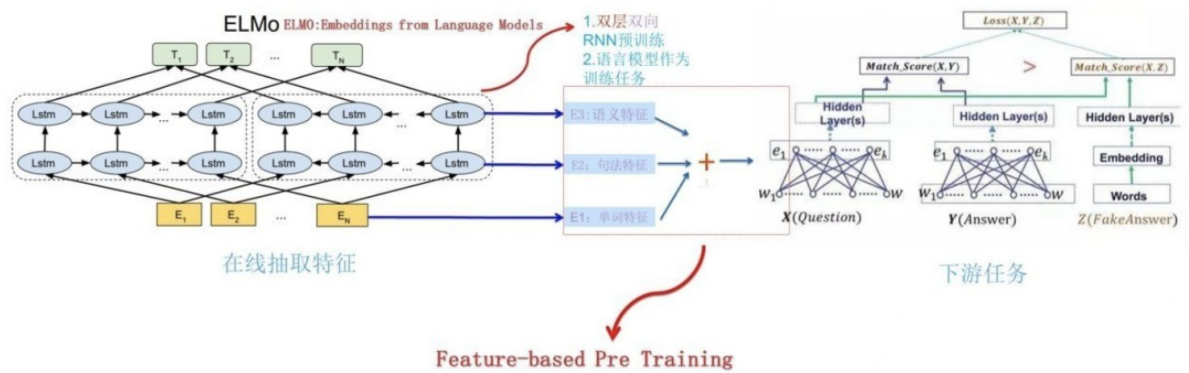
- 应该如何对这个流水线进行编码呢？感谢像 spaCy 这样神奇的 Python 库，它已经完成了！这些步骤都是编码过的，可以随时使用。

◦ Stanza库

◦ 对比

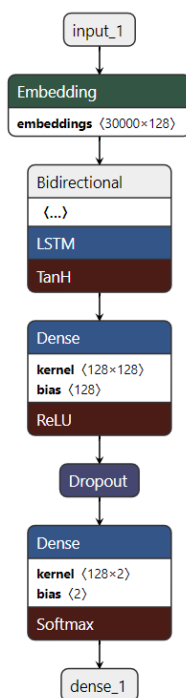
System	# Human Languages	Programming Language	Raw Text Processing	Fully Neural	Pretrained Models	State-of-the-art Performance
CoreNLP	6	Java	✓		✓	
FLAIR	12	Python		✓	✓	✓
spaCy	10	Python	✓		✓	
UDPipe	60	C++	✓		✓	✓
Stanza	66	Python	✓	✓	✓	✓

• ELMo算法



模型定义

神经网络模型架构如下所示：



训练完一个模型后，为了以后重复使用，需要对模型的结果进行保存。用 Tensorflow 去实现神经网络，所要保存的就是神经网络中的各项权重值。

四、实验结果及分析

评测指标——F1分数



NB

调参过程

```
=====alpha_: 1 --f1: 0.07492795389048991 =====
=====alpha_: 0.5 --f1: 0.253763440860215 =====
=====alpha_: 0.4 --f1: 0.34146341463414637 =====
=====alpha_: 0.3 --f1: 0.40916530278232405 =====
=====alpha_: 0.2 --f1: 0.4358208955223881 =====
=====alpha_: 0.1 --f1: 0.4555873925501433 =====
=====alpha_: 0.005 --f1: 0.4162257495590829 =====
=====alpha_: 0.001 --f1: 0.41198501872659177 =====
```

伯努利

```
=====alpha_: 1 --f1: 0.0 =====
=====alpha_: 0.5 --f1: 0.0 =====
=====alpha_: 0.4 --f1: 0.0 =====
=====alpha_: 0.3 --f1: 0.0 =====
=====alpha_: 0.2 --f1: 0.0 =====
=====alpha_: 0.1 --f1: 0.08450704225352113 =====
=====alpha_: 0.005 --f1: 0.14814814814814814 =====
=====alpha_: 0.001 --f1: 0.14016172506738542 =====
```

多项式

调参分析

伯努利模型vs多项式模型

1. 当训练样本数较少且文本单词量过少时，多项式模型中会出现大量的0或1，这个与伯努利是一样的，但是如果有极个别词出现次数特别多，就会导致概率向这个词倾斜，而伯努利因为只记录出现与否，不会记录次数，所以就不会有这个问题。
2. 但是鉴于伯努利没有将样本的词频信息纳入考量，因此在样本较多且文档较长（比如新闻分类）时，多项式模型的劣势会被削弱，优势就能体现出来
3. 因此，在训练样本数较少且文本单词量过少时，伯努利模型优于多项式模型；当样本足够丰富且文本单词量较多时，多项式模型优于伯努利模型



KNN

调参过程

```
=====n_neighbors: 114 --distanceFunction: manhattan --f1: 0.48275862068965514 =====
=====n_neighbors: 228 --distanceFunction: manhattan --f1: 0.2889518413597734 =====
=====n_neighbors: 57 --distanceFunction: manhattan --f1: 0.4274973147153598 =====
```

```
=====n_neighbors: 114 --distanceFunction: euclidean --f1: 0.10322580645161289 =====
=====n_neighbors: 114 --distanceFunction: manhattan --f1: 0.45232815964523276 =====
```

调参分析

K值

如果选择较小的K值，就相当于用**较小的领域中的训练实例进行预测**，“学习”近似误差会减小，只有与输入实例较近或相似的训练实例才会对预测结果起作用，与此同时带来的问题是“学习”的估计误差会增大，换句话说，**K值的减小就意味着整体模型变得复杂，容易发生过拟合**；

如果选择较大的K值，就相当于**用较大领域中的训练实例进行预测**，其优点是可以减少学习的估计误差，但缺点是学习的近似误差会增大。这时候，与输入实例较远（不相似的）训练实例也会对预测器作用，使预测发生错误，**且K值的增大就意味着整体的模型变得简单**。

在实际应用中，K值一般取一个比较小的数值，约为 \sqrt{N}

NB vs KNN

由于数据集本身较短的特性,一些过短评论如果用词模糊尚无法完全准确判断。而在对不同算法最优情况进行比较后得出,相较于KNN 算法,朴素贝叶斯算法有着更优的精确度,可见在处理此类问题时,朴素贝叶斯更优，但其缺点是受训练数据影响特别大，特征必须独立存在。



LR

调参过程

```
=====class_weight: None --f1: 0.38502673796791437 =====
=====class_weight: balanced --f1: 0.5952712100139081 =====
```

调参分析

class_weight参数

balanced – 类库会根据训练样本量来计算权重。某种类型样本量越多，则权重越低，样本量越少，则权重越高。

None – 默认值，不指定权重



RNN-双向LSTM

调参过程

```
=====var1_activation: relu --var2_dropoutRate: 0.4 =====
Epoch 1/3
82/82 [=====] - ETA: 0s - loss: 0.3831 - acc: 0.8792- val_f1: 0.877692
82/82 [=====] - 201s 2s/step - loss: 0.3831 - acc: 0.8792 - val_loss: 0.3749 - val_acc: 0.8777 - lr: 0.0010
lr changed to 0.0009000000427477062
Epoch 2/3
82/82 [=====] - ETA: 0s - loss: 0.3170 - acc: 0.8909- val_f1: 0.874231
82/82 [=====] - 194s 2s/step - loss: 0.3170 - acc: 0.8909 - val_loss: 0.4165 - val_acc: 0.8742 - lr: 9.0000e-04
lr changed to 0.0008100000384729356
Epoch 3/3
82/82 [=====] - ETA: 0s - loss: 0.1717 - acc: 0.9355- val_f1: 0.803846
82/82 [=====] - 194s 2s/step - loss: 0.1717 - acc: 0.9355 - val_loss: 0.5841 - val_acc: 0.8038 - lr: 8.1000e-04
训练耗时: 625.9154319763184 秒
=====var1_activation: relu --var2_dropoutRate: 0.5 =====
Epoch 1/3
82/82 [=====] - ETA: 0s - loss: 0.3830 - acc: 0.8848- val_f1: 0.877692
82/82 [=====] - 199s 2s/step - loss: 0.3830 - acc: 0.8848 - val_loss: 0.3718 - val_acc: 0.8777 - lr: 0.0010
lr changed to 0.0009000000427477062
Epoch 2/3
82/82 [=====] - ETA: 0s - loss: 0.3253 - acc: 0.8909- val_f1: 0.877692
82/82 [=====] - 191s 2s/step - loss: 0.3253 - acc: 0.8909 - val_loss: 0.4067 - val_acc: 0.8777 - lr: 9.0000e-04
lr changed to 0.0008100000384729356
Epoch 3/3
82/82 [=====] - ETA: 0s - loss: 0.2050 - acc: 0.9184- val_f1: 0.845000
82/82 [=====] - 189s 2s/step - loss: 0.2050 - acc: 0.9184 - val_loss: 0.5494 - val_acc: 0.8450 - lr: 8.1000e-04
训练耗时: 626.0499038696289 秒
```

```

=====var1_activation: relu --var2_dropoutRate: 0.6 =====
Epoch 1/3
82/82 [=====] - ETA: 0s - loss: 0.3933 - acc: 0.8795- val_f1: 0.877692
82/82 [=====] - 200s 2s/step - loss: 0.3933 - acc: 0.8795 - val_loss: 0.3728 - val_acc: 0.8777 - lr: 0.0010
lr changed to 0.0009000000427477062
Epoch 2/3
82/82 [=====] - ETA: 0s - loss: 0.3263 - acc: 0.8908- val_f1: 0.877692
82/82 [=====] - 195s 2s/step - loss: 0.3263 - acc: 0.8908 - val_loss: 0.5100 - val_acc: 0.8777 - lr: 9.0000e-04
lr changed to 0.0008100000384729356
Epoch 3/3
82/82 [=====] - ETA: 0s - loss: 0.1908 - acc: 0.9265- val_f1: 0.846923
82/82 [=====] - 193s 2s/step - loss: 0.1908 - acc: 0.9265 - val_loss: 0.5251 - val_acc: 0.8469 - lr: 8.1000e-04
训练耗时: 625.4592816829681 秒
=====var1_activation: sigmoid --var2_dropoutRate: 0.4 =====
Epoch 1/3
82/82 [=====] - ETA: 0s - loss: 0.3730 - acc: 0.8882- val_f1: 0.877692
82/82 [=====] - 199s 2s/step - loss: 0.3730 - acc: 0.8882 - val_loss: 0.3719 - val_acc: 0.8777 - lr: 0.0010
lr changed to 0.0009000000427477062
Epoch 2/3
82/82 [=====] - ETA: 0s - loss: 0.3149 - acc: 0.8907- val_f1: 0.876538
82/82 [=====] - 193s 2s/step - loss: 0.3149 - acc: 0.8907 - val_loss: 0.4225 - val_acc: 0.8765 - lr: 9.0000e-04
lr changed to 0.0008100000384729356
Epoch 3/3
82/82 [=====] - ETA: 0s - loss: 0.1728 - acc: 0.9333- val_f1: 0.863462
82/82 [=====] - 190s 2s/step - loss: 0.1728 - acc: 0.9333 - val_loss: 0.5660 - val_acc: 0.8635 - lr: 8.1000e-04
训练耗时: 625.4955966472626 秒
=====var1_activation: sigmoid --var2_dropoutRate: 0.5 =====
Epoch 1/3
82/82 [=====] - ETA: 0s - loss: 0.3722 - acc: 0.8814- val_f1: 0.877692
82/82 [=====] - 200s 2s/step - loss: 0.3722 - acc: 0.8814 - val_loss: 0.3725 - val_acc: 0.8777 - lr: 0.0010
lr changed to 0.0009000000427477062
Epoch 2/3
82/82 [=====] - ETA: 0s - loss: 0.3043 - acc: 0.8923- val_f1: 0.849231
82/82 [=====] - 190s 2s/step - loss: 0.3043 - acc: 0.8923 - val_loss: 0.4575 - val_acc: 0.8492 - lr: 9.0000e-04
lr changed to 0.0008100000384729356
Epoch 3/3
82/82 [=====] - ETA: 0s - loss: 0.1422 - acc: 0.9489- val_f1: 0.848462
82/82 [=====] - 190s 2s/step - loss: 0.1422 - acc: 0.9489 - val_loss: 0.6011 - val_acc: 0.8485 - lr: 8.1000e-04
训练耗时: 625.3822059631348 秒
=====var1_activation: sigmoid --var2_dropoutRate: 0.6 =====
Epoch 1/3
82/82 [=====] - ETA: 0s - loss: 0.3694 - acc: 0.8892- val_f1: 0.877692
82/82 [=====] - 199s 2s/step - loss: 0.3694 - acc: 0.8892 - val_loss: 0.3730 - val_acc: 0.8777 - lr: 0.0010
lr changed to 0.0009000000427477062
Epoch 2/3
82/82 [=====] - ETA: 0s - loss: 0.3143 - acc: 0.8913- val_f1: 0.870769
82/82 [=====] - 192s 2s/step - loss: 0.3143 - acc: 0.8913 - val_loss: 0.4257 - val_acc: 0.8708 - lr: 9.0000e-04
lr changed to 0.0008100000384729356
Epoch 3/3
82/82 [=====] - ETA: 0s - loss: 0.1665 - acc: 0.9384- val_f1: 0.804231
82/82 [=====] - 193s 2s/step - loss: 0.1665 - acc: 0.9384 - val_loss: 0.6259 - val_acc: 0.8042 - lr: 8.1000e-04
训练耗时: 584.0873739719391 秒
=====var1_activation: tanh --var2_dropoutRate: 0.4 =====
Epoch 1/3
82/82 [=====] - ETA: 0s - loss: 0.3926 - acc: 0.8808- val_f1: 0.877692
82/82 [=====] - 198s 2s/step - loss: 0.3926 - acc: 0.8808 - val_loss: 0.3725 - val_acc: 0.8777 - lr: 0.0010
lr changed to 0.0009000000427477062
Epoch 2/3
82/82 [=====] - ETA: 0s - loss: 0.3647 - acc: 0.8900- val_f1: 0.877692
82/82 [=====] - 190s 2s/step - loss: 0.3647 - acc: 0.8900 - val_loss: 0.3759 - val_acc: 0.8777 - lr: 9.0000e-04
lr changed to 0.0008100000384729356
Epoch 3/3
82/82 [=====] - ETA: 0s - loss: 0.2793 - acc: 0.8987- val_f1: 0.845385
82/82 [=====] - 188s 2s/step - loss: 0.2793 - acc: 0.8987 - val_loss: 0.4603 - val_acc: 0.8454 - lr: 8.1000e-04
训练耗时: 626.3545534610748 秒
=====var1_activation: tanh --var2_dropoutRate: 0.5 =====
Epoch 1/3
82/82 [=====] - ETA: 0s - loss: 0.3918 - acc: 0.8811- val_f1: 0.877692
82/82 [=====] - 194s 2s/step - loss: 0.3918 - acc: 0.8811 - val_loss: 0.3721 - val_acc: 0.8777 - lr: 0.0010
lr changed to 0.0009000000427477062
Epoch 2/3
82/82 [=====] - ETA: 0s - loss: 0.3560 - acc: 0.8908- val_f1: 0.877692
82/82 [=====] - 191s 2s/step - loss: 0.3560 - acc: 0.8908 - val_loss: 0.3718 - val_acc: 0.8777 - lr: 9.0000e-04
lr changed to 0.0008100000384729356
Epoch 3/3
82/82 [=====] - ETA: 0s - loss: 0.3479 - acc: 0.8908- val_f1: 0.877692
82/82 [=====] - 186s 2s/step - loss: 0.3479 - acc: 0.8908 - val_loss: 0.3723 - val_acc: 0.8777 - lr: 8.1000e-04
训练耗时: 570.633142709732 秒
=====var1_activation: tanh --var2_dropoutRate: 0.6 =====

```

```
Epoch 1/3
82/82 [=====] - ETA: 0s - loss: 0.4192 - acc: 0.8696- val_f1: 0.877692
82/82 [=====] - 199s 2s/step - loss: 0.4192 - acc: 0.8696 - val_loss: 0.3782 - val_acc: 0.8777 - lr: 0.0010
lr changed to 0.0009000000427477062
Epoch 2/3
82/82 [=====] - ETA: 0s - loss: 0.3539 - acc: 0.8908- val_f1: 0.877692
82/82 [=====] - 191s 2s/step - loss: 0.3539 - acc: 0.8908 - val_loss: 0.3759 - val_acc: 0.8777 - lr: 9.0000e-04
lr changed to 0.0008100000384729356
Epoch 3/3
82/82 [=====] - ETA: 0s - loss: 0.3530 - acc: 0.8908- val_f1: 0.877692
82/82 [=====] - 188s 2s/step - loss: 0.3530 - acc: 0.8908 - val_loss: 0.3720 - val_acc: 0.8777 - lr: 8.1000e-04
训练耗时: 626.0441284179688 秒
```

最佳参数为：

激活函数:tanh — dropoutRate:0.4

调参分析

1. 学习率

- 最理想的结果是使用合适的学习率来训练模型，模型的代价函数值下降得比较快，并且最后的代价函数也能够下降到一个比较小的位置，结果最理想

2. 激活函数

- 本次实验激活函数为tanh时表现较好，可能是因为：
 - relu函数训练的时候很“脆弱”，例如，一个非常大的梯度流过一个 ReLU 神经元，更新过参数之后，这个神经元再也不会对任何数据有激活现象了，那么这个神经元的梯度就永远都会是 0
 - sigmoid缺点：激活函数计算量大，反向传播求误差梯度时，求导涉及除法；反向传播时，很容易就会出现梯度消失的情况，从而无法完成深层网络的训练；Sigmoids函数饱和且kill掉梯度。
 - tanh也称为双切正切函数，取值范围为[-1,1]。tanh在**特征相差明显时**的效果会很好，如本次实验，在循环过程中会不断扩大特征效果。与 sigmoid 的区别是，tanh 是 0 均值的，因此实际应用中 tanh 会比 sigmoid 更好。

3. dropoutRate

- dropout简单来讲，就是在迭代的过程中，随机的丢弃掉某些神经元，使得其训练只包含部分神经元的网络，因为任何神经元都有可能消失，所以模型会变得对神经元不那么敏感，表现就是对参数W的压缩，起到与L2正则化类似的作用。
- 控制好dropout rate也是调参的关键，调好了就是加大模型鲁棒性，调不好就是overfitting。**一般情况，dropout rate 设为0.3-0.5即可**

评测指标——运行时间

创建朴素贝叶斯模型

使用词向量和编码标签进行朴素贝叶斯模型建立并进行预测得到概率。我们使用 scikit-learn 库创建基本模型。然后，利用 fit() 拟合函数对模型进行训练。然后，可以使用训练好的模型进行预测。

```
[8] from sklearn.naive_bayes import BernoulliNB
nb = BernoulliNB(alpha=0.03)
nb.fit(x_train,y_train) # 指定标签列为新创建的 'target' 列
predicted_labels = nb.predict(x_test)
```

创建KNN模型

使用词向量和编码标签进行 K 近邻模型建立。我们使用 scikit-learn 的 `KNeighborsClassifier` 类创建基本模型，并将 K 的值传递给模型。然后，利用 `fit()` 拟合函数对模型进行训练。然后，可以使用训练好的模型进行预测。

57秒

```
[8] from sklearn.neighbors import KNeighborsClassifier
    knn = KNeighborsClassifier(n_neighbors=3)
    knn.fit(x_train,y_train) # 指定标签列为新创建的 'target' 列
    predicted_labels = knn.predict(x_test)
```

创建LR模型

使用词向量和编码标签进行逻辑回归模型建立并进行预测得到概率。我们使用 scikit-learn 的 `LogisticRegression` 类创建基本模型。然后，利用 `fit()` 拟合函数对模型进行训练。然后，可以使用训练好的模型进行预测。

22秒

```
[8] from sklearn.linear_model import LogisticRegression
    logreg = LogisticRegression()
    logreg.fit(x_train,y_train) # 指定标签列为新创建的 'target' 列
    predicted_labels = logreg.predict(x_test)
    logreg.predict_proba(tfidf_df)[:1]
```

朴素贝叶斯分类器的训练速度比线性模型更快。这种高效率所付出的代价是，朴素贝叶斯模型的泛化能力要比线性分类器（如 LR）稍差。

朴素贝叶斯模型如此高效的原因在于，它通过单独查看每个特征来学习参数，并从每个特征中收集简单的类别统计数据。

bi_lstm_training_v4.ipynb

文件 修改 视图 插入 代码执行程序 工具 帮助 上次保存时间: 22:43

RAM 磁盘

模型训练

10分钟

```
start = time.time() # 记录训练开始时间

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          callbacks=[metrics, reduce_lr],
          validation_data=(x_test, y_test))

end = time.time() # 记录训练结束时间
print('训练耗时: ',end - start, '秒')
```

Epoch 1/3
82/82 [=====] - ETA: 0s - loss: 0.3612 - acc: 0.8868 - val_f1: 0.893462
82/82 [=====] - 191s 2s/step - loss: 0.3612 - acc: 0.8868 - val_loss: 0.3406 - val_acc: 0.8935 - lr: 0.0010
Epoch 2/3
82/82 [=====] - ETA: 0s - loss: 0.3081 - acc: 0.8892 - val_f1: 0.890385
82/82 [=====] - 190s 2s/step - loss: 0.3081 - acc: 0.8892 - val_loss: 0.3985 - val_acc: 0.8904 - lr: 0.0010
Epoch 3/3
82/82 [=====] - ETA: 0s - loss: 0.1502 - acc: 0.9477 - val_f1: 0.799615
82/82 [=====] - 189s 2s/step - loss: 0.1502 - acc: 0.9477 - val_loss: 0.5272 - val_acc: 0.7996 - lr: 0.0010
训练耗时: 621.9807171821594 秒

深度学习算法需要很长的时间来训练，这是因为在深度学习算法中有太多的参数，所以训练这些参数的时间比平时要长。

五、总结


LSTM 使我们在 RNN 中获得的重要成功。很自然地，我们也会考虑：哪里会有更加重大的突破呢？在研究人员间普遍的观点是：注意力！这个想法是让 RNN 的每一步都从更加大的信息集中挑选信息。例如：如果你使用 RNN 来产生一个图片的描述，可能会选择图片的一个部分，根据这部分信息来产生输出的词。

通过这次实验，我还了解到了Dropout可以比较有效的缓解过拟合的发生，在一定程度上达到正则化的效果。说的简单一点就是：我们在前向传播的时候，让某个神经元的激活值以一定的概率p停止工作，这样可以使模型泛化性更强，因为它不会太依赖某些局部的特征

六、参考资料

ACL2017 纽约州立大学石溪分校：基于社交媒体文本的反事实思维识别

你和“懂AI”之间，只差了一篇文章 很多读者给芯君后台留言，说看多了相对简单的AI科普和AI方法论，想看点有深度、有厚度、有眼界.....以及重口味的专业论文。为此，在多位AI领域的专家学者的帮助下，我们解读翻译了一组顶会论文。每一篇论文翻译校对完成，芯君和编辑部的老师们都会一起笑到崩溃，当然有的论文我们

 https://www.sohu.com/a/228656281_100118081




Chapter 10 Neural Network Interpretation | Interpretable Machine Learning

This chapter is currently only available in this web version. ebook and print will follow. The following chapters focus on interpretation methods for neural networks. The methods visualize features and concepts learned by a neural network, explain individual predictions and simplify neural networks.

<https://christophm.github.io/interpretable-ml-book/neural-networks.html>

ELMo算法详解_lzk_nus的博客-CSDN博客_elmo输入


ELMo来自于论文《Deep contextualized word representations》，介绍了一种高效的动态词向量。在摘要部分，作者提到词向量主要是用来解决两大问题：单词使用的复杂性，例如语法、语义不同语境下的单词使用，例如同义词 传统的Word2Vec或者Glove只能解决第一个问题，但是他们本身都是静态的词向量，也就是说每个

 https://blog.csdn.net/qq_42791848/article/details/122374703



LSTM: IndexError: Dimension out of range (expected to be in range of [-1, 0], but got 1)


Hello all, I am quite new to PyTorch so this might be quite a trivial question. I have some time-series data from 4 classes and using the CrossEntropyLoss(). I keep getting the error as mentioned in the topic. I did read a few other threads regarding this error but was not able to figure it out.

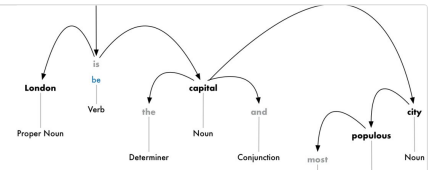
 <https://discuss.pytorch.org/t/lstm-indexerror-dimension-out-of-range-expected-to-be-in-range-of-1-0-but-go-t-1/68807>



入门 | 一步步教你构建 NLP pipeline_机器学习算法与Python学习-公众号的博客-CSDN博客

自然语言处理，或简称为 NLP，是 AI 的子领域，重点放在使计算机能够理解和处理人类语言。接下来让我们看看 NLP 是如何工作，并学习如何使用 Python 编程来从原始文本中提取信息。（注意：如果你不关心 NLP 是如何工作的，只想复制和粘贴一些代码，请跳过到「在 Python 中实现 NLP 流水线」的部分。从文本中提取

 https://blog.csdn.net/qq_28168421/article/details/87399880



Tensorflow与keras学习 (8)--实现f1_score(多分类、二分类)

keras学习：实现f1_score(多分类、二分类)本文链接：<https://blog.csdn.net/linxid/article/details/82861957>首先容易谷歌到的两种方法:1.构造metrics这种方法适用于二分类，在模型训练的时候可以作为metrics使用。使用的是固定阈值0.5。from keras import backend as Kdef...

 <https://yafeng.blog.csdn.net/article/details/102728322>




浅谈keras使用中val_acc和acc值不同步的思考_python_脚本之家

这篇文章主要介绍了浅谈keras使用中val_acc和acc值不同步的思考，具有很好的参考价值，希望对大家有所帮助。一起跟随小编过来看看吧

 <https://www.jb51.net/article/188940.htm>


Keras实现F1-score的计算

Keras只在上古版本中存在计算F1-score指标的方法，在后面的版本迭代中这些方法统统被删除掉了。原因在于，这些指标在batch-wise上计算都没有意义，需要在整个验证集上计算，而tf.keras在训练过程中（包括验证集）中计算acc、loss都是以batch为单位，最后再平均起来。然而在分类问题中，F1-score是一个很重要的评价指标，因此需要在keras中实现F1-score的计算。在网上搜寻资料的过程中，我发现其实全网解决方案无外乎3种。首先，在百度的搜索结果中，这一种结果所占的比例

 <http://www.fomalhaut.cn/370>

【机器学习】K近邻（KNN）算法详解

KNN (K Near Neighbor)：k个最近的邻居，即每个样本都可以用它最接近的k个邻居来代表。KNN算法属于监督学习方式的分类算法，我的理解就是计算某给点到每个点的距离作为相似度的反馈。简单来讲，KNN就是“近朱者赤，近墨者黑”的一种分类算法。KNN是一种基于实例的学习，属于懒惰学习，即没有显式学习过程。...

 <http://www.proyy.com/6969112765151576094.html>



