**虚假新闻检测——基于Python的信息分类问题**

**一、实验概述**

|  |
| --- |
| * 问题概述   虚假新闻在社交媒体和新闻平台上广泛存在，其对公众观念和舆论产生了深远影响。作为Python大作业，本实验旨在通过探索虚假新闻的分类方法来使我们了解机器学习算法和自然语言处理技术。我们已收集一组新闻标题样本，并利用这些数据训练模型，以实现对新闻真实性的判断。通过此实验，我们希望不仅能够对Python机器学习等方面有所了解，也能够探索到相对有效的虚假新闻检测方法。   * 实验目的   基于给定的训练集和测试集，我们来学习、构建、测试和分析相关的模型方法，从而探究更好的适用于虚假新闻检测的模型与方法。 |

**二、数据集说明**

|  |
| --- |
| 数据集名称：trian.news.csv（训练集），test.feature.csv（测试集）  数据集简介：上述两组数据集各有10000条左右的数据，每条数据均包含公名称(Official Account Name)、新闻标题(Title)、新闻内容链接(News Url)、图片链接(Image Url)、相关评论(Report Content)，其中训练集还包括每条新闻对应的id和其真实性(label为0即为真，为1即为假)，下图为示例： |

**三、方法介绍与分析**

在本次实验中，我们不难发现在给定的数据集中，News Url、Image Url以及Report Content在真假新闻上并没有很好的区别度，因此我们采用Title和Official Account Name来作为数据，其中Title为主要数据。在数据预处理部分，我们主要采用了两种方式，因此我们主要分两大块来介绍模型的整体方法。

**3.1 基于TF-IDF下的传统机器学习模型**

**3.1.1 数据预处理**

|  |
| --- |
| Python data=pd.read\_csv(r"D:/pyworks/data/train.news.csv") data\_validate=pd.read\_csv(r"D:/pyworks/data/test.feature.csv")#加载数据 x,y,x\_validate=data['Title'],data['label'],data\_validate['Title']#数据选择 vectorizer=TfidfVectorizer()#创建一个TF-IDF向量化器 x=vectorizer.fit\_transform(x)#文本向量化，将文本数据转换为特征矩阵 x\_validate=vectorizer.transform(x\_validate) x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,stratify=y,random\_state = 0) #使用 train\_test\_split 函数将训练集划分为训练集和验证集。test\_size 参数表示验证集的比例，stratify 参数确保分割后的训练集和验证集中类别分布相似。 |

整个过程的主要目的是将文本数据转换为数值特征，以便后续使用机器学习模型进行训练和评估。TF-IDF是一种常用的文本特征提取方法，通过考虑单词的重要性来帮助区分文本之间的差异。

**3.1.2 模型选择及训练**

1. **Multinomial Naive Bayes (MultinomialNB)**

* 原理： 朴素贝叶斯方法的一种，适用于多项式分布数据。
* 适用场景： 通常用于文本分类等离散特征的分类问题。

1. **K近邻（KNeighborsClassifier）**

* 原理： 基于近邻的分类算法，通过查找最接近的邻居来进行分类。
* 适用场景： 对于简单的分类问题，适用于小规模数据集。

1. **决策树（DecisionTreeClassifier）**

* 原理： 通过树状图模型进行决策，每个节点表示一个特征，每个分支代表一个决策。
* 适用场景： 可以处理分类和回归问题，对于解释性要求较高的场景较为适用。

1. **随机森林（RandomForestClassifier）**

* 原理： 由多个决策树组成，每个树进行投票，最终结果为投票最多的类别。
* 适用场景： 随机森林通过集成学习提高模型性能，适用于中大规模数据集。

1. **梯度提升（GradientBoostingClassifier）**

* 原理： 通过迭代训练多个弱学习器，每一轮都尝试修正前一轮的错误。
* 适用场景： 适用于各种数据类型，对于性能要求较高的问题效果显著。

1. **支持向量机（SVC）**

* 原理： 构建一个决策边界，使其尽可能远离训练点，以最大程度上确保分类的准确性。
* 适用场景： 适用于小到中型数据集，特别是在高维空间中的分类问题。

1. **多层感知机（MLPClassifier）**

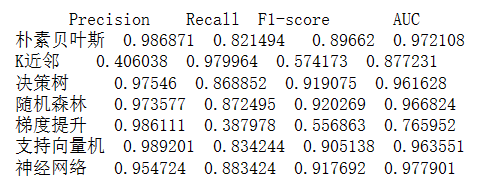
* 原理： 神经网络模型，由多个层组成，包括输入层、隐藏层和输出层。
* 适用场景： 适用于复杂的非线性关系，如图像识别和语音识别。

我们采用上述模型并调配适当的参数来进行训练。

|  |
| --- |
| Python #朴素贝叶斯 model1 = MultinomialNB() # K近邻 model2 = KNeighborsClassifier(n\_neighbors=4) # 决策树 model3 = DecisionTreeClassifier(random\_state=64) # 随机森林 model4 = RandomForestClassifier(n\_estimators=500, max\_features='sqrt', random\_state=10) # 梯度提升 model5 = GradientBoostingClassifier(random\_state=128) # 支持向量机 model6 = SVC(kernel="rbf", random\_state=64) # 神经网络（多层感知机） model7 = MLPClassifier(hidden\_layer\_sizes=(16, 8), random\_state=64, max\_iter=10000) |

**3.1.3 实验结果及分析**

实验结果如下图所示(在训练集划分出来的的测试集上运行的结果)：



我们不难看出，在基于词频的TF-IDF方法提出的特征中，我们将整个Title直接作为输入时，K近邻的准确率和梯度提升的召回率很低，效果不佳，而其他的模型也仍有改进的空间。经过大量资料的查找和与他人的学习交流后，笔者认为可以使用jieba分词、改进模型等方式来优化整个方法，但笔者并未花太多时间在此之中，而是投入到接下来的第二点。

**3.2 基于Word2Vec的深度学习模型**

**3.2.1 数据预处理**

Word2Vec 是一种常用于自然语言处理的词嵌入技术，它可以将单词映射到一个连续的向量空间中，在处理短文本(新闻标题)中，它可以通过计算单词之间的向量相似性来衡量它们之间的语义关联性。我们将每个单词的向量表示进行平均或拼接，可以得到整个短文本的向量表示，从而帮助提高分类的准确性。此外，由于Word2Vec 可以提供单词级别的语义信息，可以帮助模型更好地理解短文本的含义，进行文本标签识别等任务。

**3.2.1.1 导入已经训练好的词向量**

|  |
| --- |
| Python train\_data = pd.read\_csv(r'D:/pyworks/data/train.news.csv') test\_data = pd.read\_csv(r"D:/pyworks/data/test.feature.csv") word2vec = open("D:/pyworks/data/sgns.sogounews.bigram-char", "r", encoding='UTF-8') t = word2vec.readline().split() wordAndVec = word2vec.readlines() wordAndVec = [i.split() for i in wordAndVec] vectorsMap = [] word2index = {} index2word = {} for i in range(n):  vectorsMap.append(list(map(float, wordAndVec[i][len(wordAndVec[i]) - dimension:])))  word2index[wordAndVec[i][0]] = i  index2word[i] = wordAndVec[i][0] word2vec.close() |

**3.2.1.2 分词与词向量构建**

|  |
| --- |
| Python features\_train = [] for title in train\_data['Title']:  word\_feature = []  for word in jieba.analyse.extract\_tags(title,topK=59,withWeight=False,allowPOS=()):  if word in word2index: word\_feature.append(vectorsMap[word2index[word]])  features\_train.append(word\_feature) |

词向量构建完成之后还需对其进行填充和转换，以使其维度数和维度大小相同，同时标签也要进行相应的处理。相关代码在此不做赘述。

**3.2.2 模型构建及训练**

此次实验我们主要采用了LSTM，辅之以CNN来构建模型，一共搭建出2种模型。

* LSTM模型

|  |
| --- |
| Python # 构建深度学习模型 model = Sequential() model.add(LSTM(128, input\_shape=(X\_train.shape[1], X\_train.shape[2]), return\_sequences=True)) model.add(SpatialDropout1D(0.5)) model.add(LSTM(64)) model.add(Dense(2, activation='softmax')) |

* LSTM+CNN模型

|  |
| --- |
| Python #一层LSTM  model = Sequential() # LSTM层 model.add(LSTM(128, input\_shape=(X\_train.shape[1], X\_train.shape[2]), return\_sequences=True)) model.add(SpatialDropout1D(0.5)) # 添加第一层卷积层 model.add(Conv1D(filters=64, kernel\_size=2, activation='relu')) model.add(MaxPooling1D()) # 添加第二层卷积层 model.add(Conv1D(filters=32, kernel\_size=2, activation='relu')) model.add(MaxPooling1D()) # Flatten层用于将卷积层的输出展平 model.add(Flatten()) # 密集连接层 model.add(Dense(2, activation='softmax')) |

|  |
| --- |
| Python #两层LSTM  model = Sequential() # 添加第一层LSTM model.add(LSTM(64, return\_sequences=True, input\_shape=(X\_train.shape[1], X\_train.shape[2]))) model.add(SpatialDropout1D(0.5)) # 添加第二层卷积层 model.add(Conv1D(filters=32, kernel\_size=3, activation='relu')) model.add(MaxPooling1D()) # 添加第三层LSTM model.add(LSTM(128, return\_sequences=True)) model.add(SpatialDropout1D(0.5)) # 添加第四层卷积层 model.add(Conv1D(filters=16, kernel\_size=3, activation='relu')) model.add(MaxPooling1D()) # Flatten层用于将卷积层的输出展平 model.add(Flatten()) # 密集连接层 model.add(Dense(2, activation='softmax')) |

模型构建后再进行训练和评估，大致代码如下：

|  |
| --- |
| Python loss, accuracy = model.evaluate(X\_validation, Y\_validation) print(f"Validation Loss: {loss}, Validation Accuracy: {accuracy}") # 计算召回率、AUC和F1分数 y\_validation\_pred = model.predict(X\_validation) y\_validation\_pred\_binary = np.argmax(y\_validation\_pred, axis=1) y\_validation\_true = np.argmax(Y\_validation, axis=1) recall = recall\_score(y\_validation\_true, y\_validation\_pred\_binary) auc\_score = roc\_auc\_score(Y\_validation, y\_validation\_pred) f1score = f1\_score(y\_validation\_true, y\_validation\_pred\_binary) print(f"Recall: {recall}, AUC: {auc\_score}, F1 Score: {f1score}") # 可视化训练过程 plt.plot(history.history['accuracy']) plt.plot(history.history['val\_accuracy']) plt.title('Model Accuracy') plt.ylabel('Accuracy') plt.xlabel('Epoch') plt.legend(['Train', 'Validation'], loc='upper left') plt.show() print(f"Recall: {recall}, AUC: {auc\_score}, F1 Score: {f1score}") |

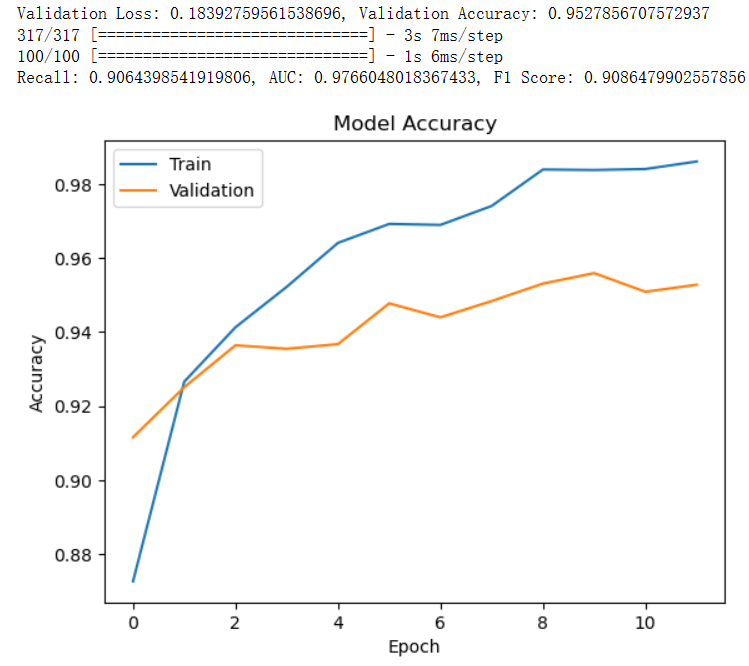
**3.2.3 实验结果及分析**

运行截图(以LSTM模型为例)：

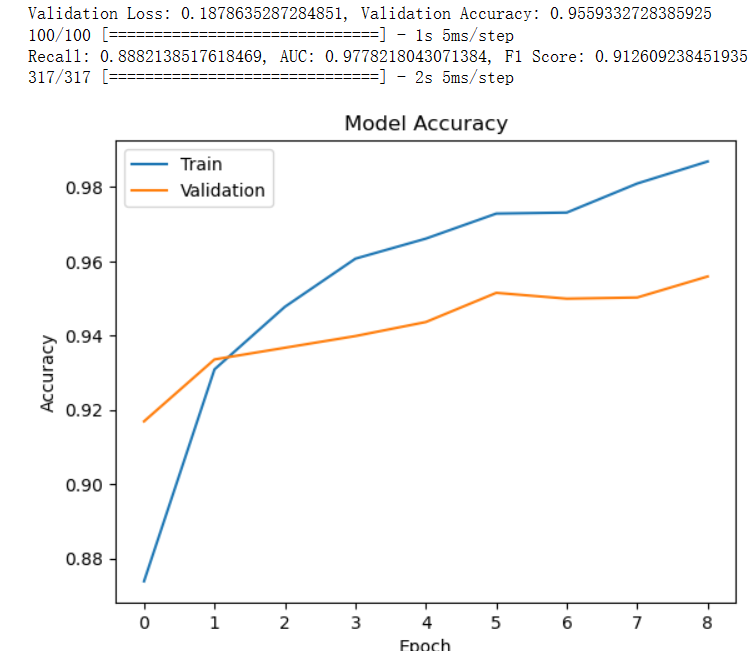


实验结果：

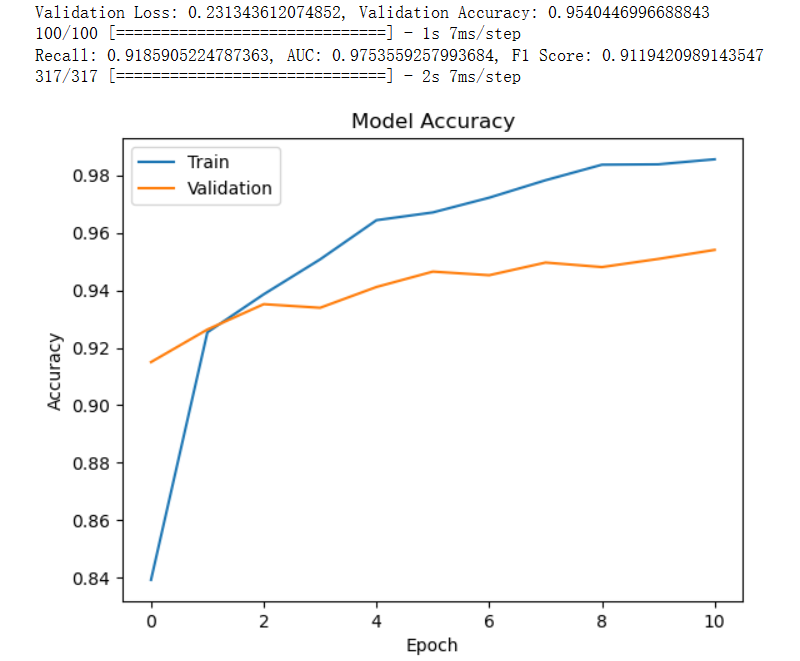
1. LSTM：



1. LSTM(一层)+CNN：



1. LSTM(两层)+CNN：



从实验结果上看，LSTM模型本身准确率就已经达到了95%以上，相应的召回率、F1 score等值也达到不错的数值。我们不难发现，加入CNN后模型并没有优化多少。能达到这种优化程度主要靠以下几个方面：

1. 词向量构建：在词向量训练方面个人觉得由于数据量较少，自己难以训练出足够且有效的数据，因而使用sogo新闻训练词向量表。在分词部分，笔者原先采用的是jieba分词，发现效果并不是很好，而且每个标题构建的词向量长度不一且差距较大，填充至相同维度时会导致某些特征矩阵过于稀疏。因此采用jieba中的TF-IDF分法，在剔除一些相对无用的数据的同时也能够对划分得到的词进行重新排序，可能更适合于LSTM模型的输入。
2. 模型构建的参数调整：对于训练样本相对少的情况，笔者认为模型中相应的超参数要调小一些。经过多次的调整和训练，笔者发现模型相关参数如上述代码所示时模型准确率较高。

以LSTM模型为例，部分参数调试过程如下图所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 丢弃率 | 隐藏层 | 1:128  2:128 | 1：128  2：64 | 1:64  2:64 | 1：64  2:128 |
| 0.5 | Acc | 0.944 | 0.951 | 0.945 | 0.946 |
|  | F1-Score | 0.885 | 0.906 | 0.896 | 0.893 |
|  | Recall | 0.836 | 0.914 | 0.911 | 0.904 |
| 0.3 | Acc | 0.952 | 0.952 | 0.951 | 0.941 |
|  | F1-Score | 0.908 | 0.907 | 0.907 | 0.882 |
|  | Recall | 0.900 | 0.908 | 0.920 | 0.840 |

经过在一定范围内的不断的调参，我们不难发现LSTM模型中两层LSTM各自的隐藏层数分别为128和64，丢弃率为0.5时模型平均准确率较好，在竞赛平台上的预测成绩最好能够达到0.84以上。相应的LSTM(一层)+CNN和LSTM(两层)+CNN模型的超参数如上述代码所示那般，基于这些值，模型也能够抛出不错的效果。

1. 概率界定值的选取：上述经过训练后的模型进行预测后会产生两个相加为1的概率值，我们需要选定合适的界定数来划分0和1，这个也是模型预测准确率提高的一个重点。

**四、其他的尝试**

笔者在构建上述模型的时候也曾尝试其他的组合，比如在数据预处理这块，尝试过将公众号名称加入进来使用去停用词后用Tokenizer等方式进行词向量的转换，但是发现使用去停用词后再用jieba分词等时，句子划分得到的有效单词数量并不如加入预训练词表后得到的数量多，故效果一般。

|  |
| --- |
| Python def extract\_chinese(content):  content\_str = ''  for i in content:  if (i >= '\u4e00' and i <= '\u9fa5') :  content\_str += i  return content\_str file=open('D:/pyworks/data/stopwords.txt', 'r',encoding='utf8') Stop\_words=[i.strip() for i in file.readlines()] file.close() data\_train=pd.read\_csv(r"train.news.csv") data\_val=pd.read\_csv(r"test.feature.csv") all\_words\_train = [] all\_words\_test = [] words\_test=[] all\_labels = [] for title,lab in zip(data\_train['Title'],data\_train['label']):  content=extract\_chinese(title)  words=lcut(content)  label=lab  words = [i for i in words if not i in Stop\_words]  if len(words)<=0:  continue  all\_words\_train.append(words)  all\_labels.append(label) for title in data\_val['Title']:  content=extract\_chinese(title)  words=lcut(content)  words = [i for i in words if not i in Stop\_words]  all\_words\_test.append(words)  words\_test.append(title) X\_train, X\_v, train\_y, v\_y = train\_test\_split(all\_words\_train,all\_labels,test\_size=0.3, random\_state=42) X\_val, X\_test, val\_y, test\_y = train\_test\_split(X\_v ,v\_y,test\_size=0.5, random\_state=42) ohe = OneHotEncoder() train\_y = ohe.fit\_transform(np.array(train\_y).reshape(-1,1)).toarray() val\_y = ohe.transform(np.array(val\_y).reshape(-1,1)).toarray() test\_y = ohe.fit\_transform(np.array(test\_y).reshape(-1,1)).toarray() |

模型构建上尝试过说单独的CNN模型，但是笔者发现它模型训练时准确率很高，但在预测时AUC并不高(不到0.8)，这也可能是跟词向量的构建有所关系。

|  |
| --- |
| Python def cnn(X\_train):  model = tf.keras.Sequential([  tf.keras.layers.Convolution1D(input\_shape=(X\_train.shape[1], X\_train.shape[2]),  filters=128, kernel\_size=3, activation='relu'),  tf.keras.layers.MaxPool1D(pool\_size=2),  tf.keras.layers.Convolution1D(128, 3, activation='relu'),  tf.keras.layers.MaxPool1D(),  tf.keras.layers.Convolution1D(64, 5),  tf.keras.layers.Dropout(rate=0.5),  tf.keras.layers.Dense(2, activation='softmax'),  ])  print(model.summary())  return model |

也尝试过Bert模型，但由于参数配置和模型理解的不够深入，以及对于pytorch的了解不多，导致我对其模型的掌握程度不是很好，因而预测效果不是很好。

**五、总结与展望**

本次实验是在给定的训练集中进行数据的处理，构建适合的模型和根据模型训练后得到的参数指标对模型进行评价和分析。笔者是第一次接触这种NLP问题的解决。经过一段时间的学习，笔者主要用Word2Vec、LSTM、CNN等方法进行问题的处理，并取得了不错的效果。当然，由于个人精力的有限和知识的匮乏，其他特征的运用和输入，例如情感分析，图片Url和新闻内容Url等并未使用，模型这块也没有将Bert模型很好的构建起来，这些是日后多加学习的。