2025

应用深度学习预测新闻标题的讽刺性

主讲人：陈德福

时间：2025.6

DESIGN

目录

CONTENT

YOUR LOGO

Part 01

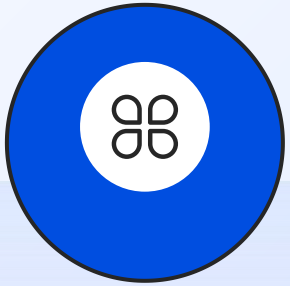
项目背景与目标

202X

项目背景

讽刺检测的重要性

随着互联网信息爆炸式增长，新闻平台的头条内容成为舆论传播的核心载体。讽刺性新闻标题常通过反语、夸张、双关等手法隐含真实意图，但其语义的隐蔽性导致传统文本分析技术难以精准识别。
新闻标题的讽刺性高度依赖语境、标点符号及领域术语，传统的机器学习方法难以捕捉多维度语义特征，导致检测准确率普遍较低。



数据来源与挑战

数据来源于网络新闻标题，具有短文本、噪声多特点。需处理数据不平衡、词向量嵌入等挑战。
使用Kaggle Sarcasm_Headlines_Dataset_v2.json数据集，包含大量新闻标题。数据集标注了是否为讽刺，适合用于监督学习。

数据描述：

1. 采用kaggle公开的Sarcasm Headlines Dataset(包含 3.6 万条标注数据),涵盖新闻标题文本及其讽刺标签(1 = 讽刺,0 = 非讽刺)。
2. 数据划分:按 8:1:1 比例划分为训练集(28,813 条)、验证集(3,602 条)、测试集(3,601 条)。
3. is_sarcastic: 一个整数值(0 或 1),表示标题是否具有讽刺性。1 表示标题是讽刺性的,0 表示标题不是讽刺性的。
4. headline: 新闻标题的文本内容。
5. article_link: 与标题相关的文章链接。

YOUR LOGO

Part 02

数据预处理

202X

数据加载与清洗

Spark数据加载

使用Spark加载大规模JSON格式数据，去除无关字段，提高数据处理效率，为后续分析奠定基础。

文本预处理

对文本进行分词、去除停用词等操作，保留关键信息，减少噪声干扰，提升模型性能。

01

- **详细过程：**
- 例:"headline": "trump's obsession with chinese currency manipulation is sooo 2014"
- 删除无关字段 :使用 df.drop('article_link') 删除 article_link 列,因为它对模型训练没有帮助。
  分词 :使用 Tokenizer 将 headline 字段中的文本拆分为单词列表。
  输入列是 headline,输出列是 words。
  去停用词:使用 StopWordsRemover 去除常见的停用词(如 "the", "is", "and" 等)。输入列是 words,输出列是 filtered_words。
  构建 Pipeline :将分词和去停用词的操作组合成一个 Spark ML Pipeline,以便高效地对数据进行处理。

词向量嵌入

GloVe词向量加载

GloVe 词向量： GloVe是一种基于词频统计的词向量表示方法，它通过构建词共现矩阵来捕捉单词之间的全局语义信息。GloVe模型可以将单词映射到一个固定维度的向量空间中，使得语义相似的单词在向量空间中的位置更接近。在本项目中，我们使用预训练的 GloVe 词向量作为基础词嵌入，其优势在于：

加载预训练的GloVe词向量，为模型提供丰富的语义信息，增强对文本的理解能力。

Word2Vec模型训练

网络新闻标题具有其独特的语言风格和词汇用法，通过在我们自己的数据集上训练 Word2Vec 模型，可以使词向量更好地适应这个特定的领域，捕捉到新闻标题中特有的语义信息和词汇关联。

使用Word2Vec对特定数据集训练词向量，捕捉数据独特语义特征，与GloVe结合提升嵌入效果。

组合嵌入矩阵构建

将GloVe和Word2Vec词向量组合，构建双倍维度嵌入矩阵，充分利用两种词向量优势，优化模型输入。

拼接融合：将 GloVe（100 维）与 Word2Vec（100 维）按维度拼接，形成 200 维词嵌入向量，综合通用语义与领域上下文信息。

特征提取与向量化：

训练 Word2Vec 模型 :利用训练集中的文本数据训练 Word2Vec 模型,将每个单词映射到一个低维向量空间中,生成能够反映单词语义信息的词向量。

加载 GloVe 词向量 :加载预训练好的 GloVe 词向量,它提供了另一种将单词转换为固定维度向量的方法,能够捕捉单词之间的语义关系。

构建组合 Embedding 矩阵 :将 Word2Vec 模型和 GloVe 词向量结合起来,创建一个组合的词向量矩阵,用于嵌入层的初始化。这样可以综合利用两种词向量的优势,提高模型对文本的理解能力。

GloVe词向量加载

```python
# 加载 GloVe 词向量
def load_glove_embeddings(path, embedding_dim=100):
    embeddings_index = {}
    with open(path, encoding='utf8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
    print(f"Loaded {len(embeddings_index)} GloVe vectors.")
    return embeddings_index
```

Word2Vec模型训练

```python
# 训练 Word2Vec 模型
def train_word2vec(texts, vector_size=100, window=5, min_count=2):
    tokenized_texts = [text.split() for text in texts]
    model = Word2Vec(sentences=tokenized_texts, vector_size=vector_size,
                     window=window, min_count=min_count, workers=4, sg=1)
    print("Word2Vec model trained.")
    return model
```

将GloVe和Word2Vec词向量组合：

```python
def create_combined_embedding_matrix(word_index, glove_path, w2v_model, embedding_dim=100):
    glove_embeddings = load_glove_embeddings(glove_path, embedding_dim)

    num_words = min(10000, len(word_index) + 1)
    embedding_matrix = np.zeros((num_words, embedding_dim * 2))  # 双倍维度

    hits = 0
    misses = 0

    for word, i in word_index.items():
        if i >= num_words:
            continue

        w2v_vector = None
        if word in w2v_model.wv:
            w2v_vector = w2v_model.wv[word]

        glove_vector = None
        if word in glove_embeddings:
            glove_vector = glove_embeddings[word]

        if w2v_vector is not None and glove_vector is not None:
            embedding_matrix[i] = np.concatenate([w2v_vector, glove_vector])
            hits += 1
        else:
            vec = w2v_vector if w2v_vector is not None else glove_vector
            if vec is not None:
                embedding_matrix[i] = np.concatenate([vec, vec])  # 填充双份
                hits += 1
            else:
                misses += 1

    print(f"Found matches for {hits} words, missed {misses}.")
    return embedding_matrix
```

YOUR LOGO

Part 03

模型构建与训练

202X

模型构建：

嵌入层 :使用预训练好的组合词向量矩阵初始化嵌入层,将文本序列中的单词转换为对应的词向量,形成文本的向量表示。

双向 LSTM 层(BiLSTM) :在嵌入层的基础上构建双向 LSTM 层,BiLSTM 可以同时考虑文本的前后语境信息,更好地捕捉文本的序列依赖关系,提取文本的深层语义特征。

注意力层(AttentionLayer) :在 BiLSTM 层后添加自定义的注意力层,该层可以自动学习文本序列中每个位置的重要性权重,使模型能够更加关注与任务相关的部分,进一步提升模型对关键信息的捕捉能力。

全连接层与输出层 :通过全连接层对提取到的特征进行整合和非线性变换,然后通过输出层输出最终的分类结果,即判断文本是否为讽刺文本。

模型架构设计

壹

BiLSTM与Attention机制

BiLSTM： 负责捕捉输入序列的上下文信息，提供丰富的特征表示。

Attention： 动态分配权重，聚焦于输入序列中的关键部分，提升模型的表达能力。

协同工作： BiLSTM 提供全局上下文，Attention 提取局部关键信息，二者结合能够显著提升模型的性能，特别是在讽刺性检测等复杂任务中。

贰

自定义Attention层实现

自定义Attention层，灵活调整模型对不同文本片段的关注程度，提高模型对复杂文本的适应性。我们引入了自定义的 Attention 层。其基本原理是为文本序列中的每个单词计算一个注意力权重，然后根据这些权重对单词向量进行加权求和，使得模型更加关注重要的词汇，从而提高模型的性能。

自定义注意力机制层：

```python
# 自定义 Attention Layer
class AttentionLayer(Layer):
    def __init__(self, **kwargs):
        super(AttentionLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        self.W = self.add_weight(name="att_weight", shape=(input_shape[-1], 1),
                                 initializer="normal")
        self.b = self.add_weight(name="att_bias", shape=(input_shape[1], 1),
                                 initializer="zeros")
        super(AttentionLayer, self).build(input_shape)

    def call(self, x):
        e = K.tanh(K.dot(x, self.W) + self.b)
        a = K.softmax(e, axis=1)
        output = x * a
        return K.sum(output, axis=1)

    def compute_output_shape(self, input_shape):
        return (input_shape[0], input_shape[-1])
```

结合BiLSTM和注意力机制：

```python
# 构建带 Attention 的 BiLSTM 模型
def build_attention_bilstm_model(embedding_matrix, maxlen=50, embedding_dim=200):
    inputs = Input(shape=(maxlen,))

    embedding = Embedding(
        input_dim=embedding_matrix.shape[0],
        output_dim=embedding_dim,
        weights=[embedding_matrix],
        trainable=False
    )(inputs)

    lstm_out = Bidirectional(LSTM(128, return_sequences=True))(embedding)
    attention_out = AttentionLayer()(lstm_out)

    dense = Dense(64, activation='relu')(attention_out)
    dropout = Dropout(0.5)(dense)
    outputs = Dense(1, activation='sigmoid')(dropout)

    model = Model(inputs=inputs, outputs=outputs)
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model
```

分布式训练

01

Spark分布式训练环境搭建

利用Spark搭建分布式训练环境，合理配置资源，提高模型训练效率，缩短训练时间。

02

分布式训练流程

将数据分片并分发到不同节点，每个节点独立训练模型，最后汇总更新权重，实现高效分布式训练。

Spark分布式训练环境搭建：

appName ("DistributedSarcasmTraining")	设置当前 Spark 应用程序的名称，在 Spark UI 或日志中可见，便于识别任务。
spark.executor.memory="4g"	每个 Executor 分配的堆内存为 4GB。Executor 是运行在 Worker 节点上的进程，负责执行任务。
spark.driver.memory="4g"	Driver 程序使用的内存为 4GB。Driver 是主控节点，负责调度任务和聚合结果。
spark.executor.cores="1"	每个 Executor 使用 1 个 CPU 核心来并行执行任务。
spark.num.executors="2"	总共启动 2 个 Executor 来执行任务。

```python
def setup_spark_for_distributed_training():
    spark = SparkSession.builder \
        .appName("DistributedSarcasmTraining") \
        .config("spark.executor.memory", "4g") \
        .config("spark.driver.memory", "4g") \
        .config("spark.executor.cores", "1") \
        .config("spark.num.executors", "2") \
        .getOrCreate()
    sc = spark.sparkContext
    print(f"Using Python executable: {sc.pythonExec}")
    return spark, sc
```

模型训练与优化

类别权重平衡：计算训练集中每个类别的权重，用于在训练过程中平衡不同类别之间的损失贡献，缓解类别不平衡问题对模型性能的影响。
编译模型：使用 Adam 优化器和二元交叉熵损失函数编译模型。Adam 优化器是一种自适应学习率优化算法，能够自动调整学习率，提高模型的收敛速度；二元交叉熵损失函数适用于二分类问题，用于衡量模型预测值与真实值之间的差异。
训练模型：使用训练集对模型进行训练，同时在每个 epoch 结束后在验证集上进行验证，以便监控模型的训练状态和性能表现。
早停：设置早停回调函数，当验证集上的损失在连续若干个 epoch 内没有改善时，提前停止训练，防止模型过拟合。
学习率衰减：设置学习率衰减回调函数，当验证集上的损失在一定 epoch 内没有显著下降时，自动降低学习率，使模型能够在训练后期更精细地调整参数，进一步提高模型的性能。

主训练程序：

```python
# 回调函数（早停）
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=1e-6)

# 类别权重平衡
class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weights_dict = dict(enumerate(class_weights))


history = model.fit(X_train_pad, y_train,
                    epochs=15,
                    batch_size=64,
                    validation_data=(X_test_pad, y_test),
                    class_weight=class_weights_dict,
                    callbacks=[early_stop, reduce_lr])

# 预测和评估
y_proba = model.predict(X_test_pad)
y_pred = (y_proba > 0.5).astype(int)

print(classification_report(y_test, y_pred, target_names=["非讽刺", "讽刺"]))
print(f"ROC AUC Score: {roc_auc_score(y_test, y_proba.ravel()):.4f}")
```

YOUR LOGO

Part 04

模型评估与结果分析

202X

模型性能评估

分类指标分析

使用准确率、召回率、F1值等指标评估模型性能，全面衡量模型对讽刺和非讽刺文本的识别能力。

ROC AUC评估

计算ROC AUC值，评估模型在不同阈值下的性能，直观反映模型对正负样本的区分能力。

模型评估与结果分析

```
358/358 [==============================] - 23s 56ms/step - loss: 0.5333 - accuracy: 0.7338 - val_loss: 0.5023 - val
Epoch 2/15
358/358 [==============================] - 20s 56ms/step - loss: 0.4590 - accuracy: 0.7869 - val_loss: 0.4488 - val
Epoch 3/15
358/358 [==============================] - 19s 53ms/step - loss: 0.4174 - accuracy: 0.8083 - val_loss: 0.4784 - val
Epoch 4/15
358/358 [==============================] - 19s 53ms/step - loss: 0.3834 - accuracy: 0.8280 - val_loss: 0.4066 - val
Epoch 5/15
358/358 [==============================] - 20s 55ms/step - loss: 0.3527 - accuracy: 0.8423 - val_loss: 0.4212 - val
Epoch 6/15
358/358 [==============================] - 20s 55ms/step - loss: 0.3194 - accuracy: 0.8618 - val_loss: 0.4107 - val
Epoch 7/15
358/358 [==============================] - 20s 55ms/step - loss: 0.2529 - accuracy: 0.8957 - val_loss: 0.4296 - val
179/179 [==============================] - 3s 12ms/step
              precision    recall  f1-score   support

         非讽刺       0.81      0.82      0.82      2995
          讽刺       0.80      0.79      0.80      2729

    accuracy                           0.81      5724
   macro avg       0.81      0.81      0.81      5724
weighted avg       0.81      0.81      0.81      5724

ROC AUC Score: 0.8944
```

模型训练过程：

训练损失：从0.5333逐渐下降到0.2529，表明模型在训练集上逐渐学习到了数据的规律，拟合能力增强。
训练准确率：从0.7338上升到0.8957，说明模型在训练集上的分类正确率逐渐提高。
验证损失：从0.5023下降到0.4296，表明模型在验证集上的表现也在改善，没有出现明显的过拟合现象。
验证准确率：从0.7419上升到0.8313，显示模型在验证集上的分类能力有所提升。

分类报告：

非讽刺（Negative）类别：
精确率（Precision）：0.81，表示模型预测为非讽刺的样本中，有81%实际是非讽刺的。
召回率（Recall）：0.82，表示实际为非讽刺的样本中，有82%被模型正确分类。
F1值：0.82，综合了精确率和召回率的调和平均。
讽刺（Positive）类别：
精确率：0.80，表示模型预测为讽刺的样本中，有80%实际是讽刺的。
召回率：0.79，表示实际为讽刺的样本中，有79%被模型正确分类。
F1值：0.80，综合精确率和召回率的平均值。

结果可视化

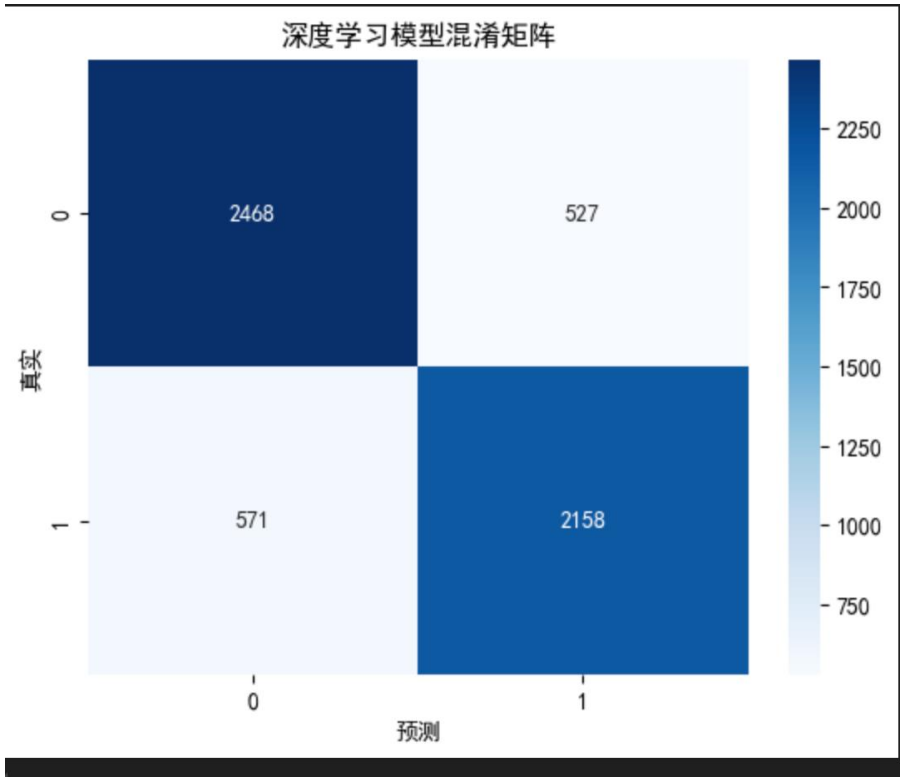
混淆矩阵绘制

绘制混淆矩阵，直观展示模型预测结果与真实标签的匹配情况，发现模型的误分类问题。

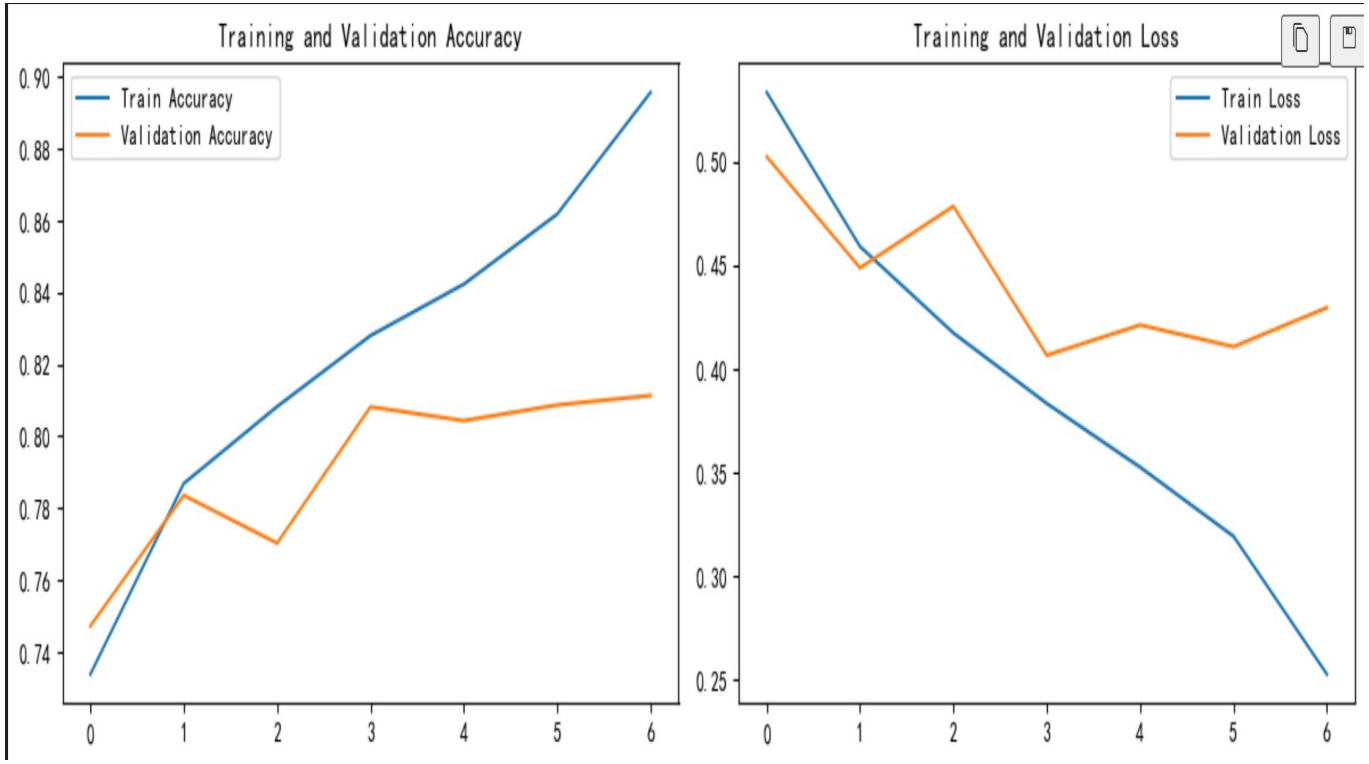
训练历史可视化

绘制训练和验证过程中的准确率、损失曲线，观察模型训练过程中的性能变化，判断模型是否过拟合或欠拟合。

混淆矩阵



训练历史可视化



结果分析:

- 数据分布与类别平衡 :从混淆矩阵可以看出,训练集和测试集中的两类(非讽刺和讽刺)样本数量相对均衡,模型在两类上的分类性能较为接近,没有明显的偏向性。
- 模型性能指标 :模型在测试集上的准确率为 0.81,ROC AUC 评分为 0.8944,表明模型在区分非讽刺和讽刺两类上有较好的性能。从分类报告中的率精确、召回率和 F1 分数来看,非讽刺类别的精确率为 0.81,召回率为 0.82,F1 分数为 0.82;讽刺类别的精确率为 0.80,召回率为 0.79,F1 分数为 0.80,说明模型在两类上的分类能力较为均衡,没有明显的过拟合某一类的情况。
- 训练过程 :从训练日志中可以看出,模型的训练集和验证集的损失逐渐下降,准确率逐渐上升,表明模型在不断学习和优化。在训练过程中,早停和学习率衰减的回调函数能够有效地防止过拟合和加快模型的收敛速度,使模型在有限的训练时间内达到较好的性能。

YOUR LOGO

Part 05

项目总结

202X

项目总结

项目概述

本项目聚焦于网络新闻标题的讽刺性检测，旨在开发一个基于深度学习的高效检测系统，以应对网络信息爆炸时代新闻标题误导性增强的挑战。我们使用了深度学习、自然语言处理技术，借助大规模数据集进行模型训练，以实现对新闻标题讽刺性的精准判断。

模型构建与创新点

我们构建了基于 BiLSTM 和自定义 Attention 机制的模型。BiLSTM 能够有效利用序列数据的前后向信息，捕捉标题中词汇的长距离依赖关系；Attention 机制则使模型聚焦于关键词汇。此外，还采用了混合词嵌入方法，结合 GloVe 和 Word2Vec 优势，并引入自定义 Attention 层，提升了模型对讽刺性语言的理解能力。

YOUR LOGO

2025

谢谢大家

DESIGN