YOUR LOGO

DESIGN

基于深度学习与Spark的讽刺检测模型

目录

CONTENT

YOUR LOGO

PART 01

项目背景与数据预处理

项目背景

讽刺检测的重要性

在社交媒体中，讽刺信息可能引发误解，准确检测有助于信息管理。

讽刺检测可应用于舆情监控，帮助企业或机构更好地理解公众情绪。

数据来源与特点

使用Kaggle的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。

数据预处理流程

清洗文本，去除URL、标点符号和数字，降低噪声。

使用NLTK进行分词和去除停用词，提取有效文本特征。

1.数据预处理：

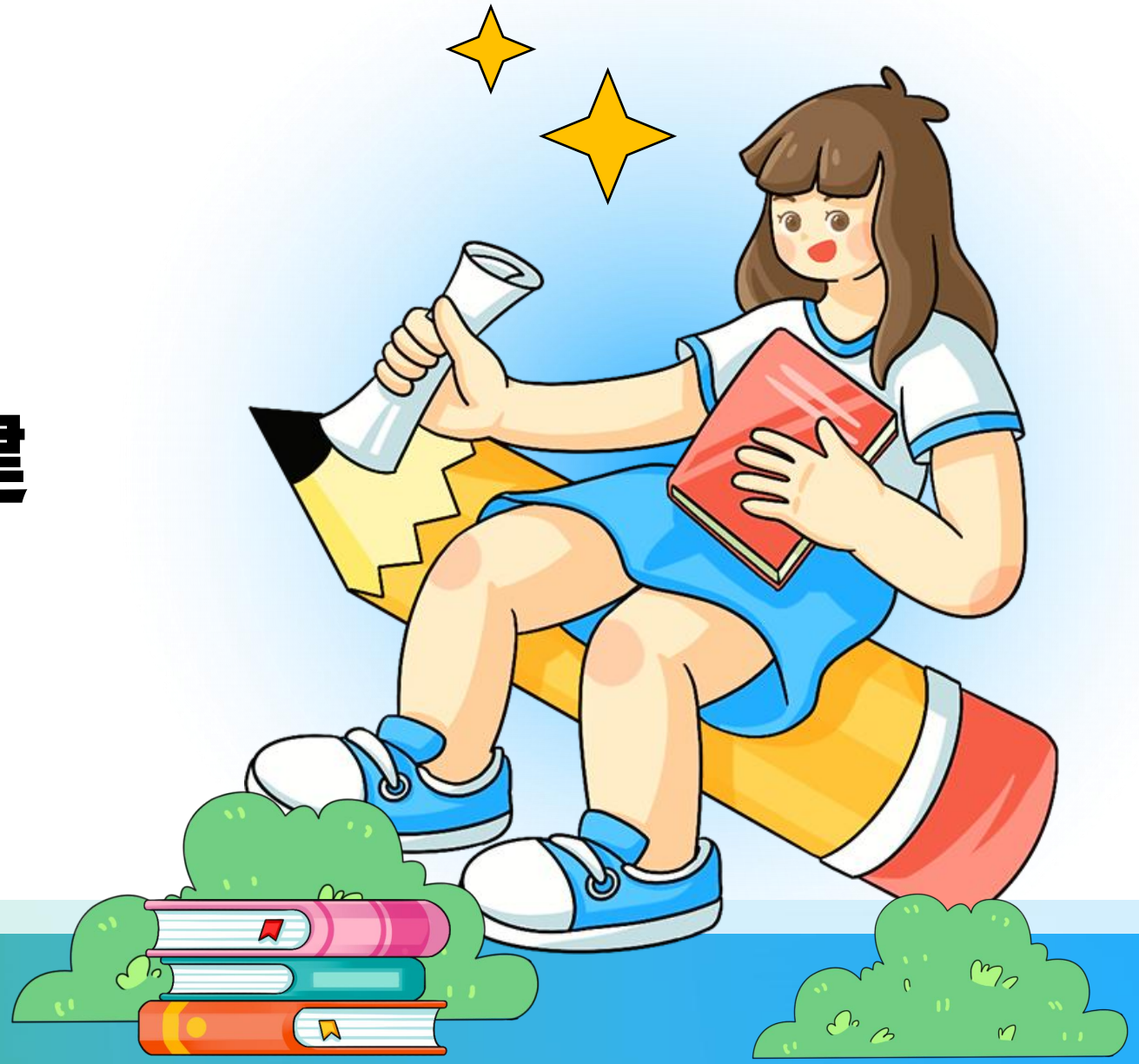
加载数据： 从 JSON 文件中读取数据，存储为 DataFrame。
删除无用列： 去掉不相关的字段（如文章链接）。
清理文本： 将标题转换为小写，去除 URL、标点符号、数字等干扰信息。
分词与去停用词： 将清理后的文本拆分为单词，并过滤掉无意义的停用词（如 the, is 等）。
返回结果： 生成一个包含清理后文本和分词结果的 DataFrame，供后续分析使用。
总结来说，这个函数的作用是把原始数据处理成干净、结构化的格式，方便后续进行自然语言处理任务。

```python
# 数据加载和预处理函数
def load_and_preprocess_data():
    df = pd.read_json(r"C:/Users/cdf/PycharmProjects/SparkRdd/kaggle/archive/Sarcasm_Headlines_Dataset
    df.drop('article_link', axis=1, inplace=True)

    def clean_text(text):
        text = text.lower()
        text = re.sub(r'http\S+', '', text)
        text = re.sub(r'\[.*?\]', '', text)
        text = re.sub(r'[%s]' % re.escape(string.punctuation), '', text)
        text = re.sub(r'\w*\d\w*', '', text)
        return text

    df['headline'] = df['headline'].apply(clean_text)

    stop_words = set(stopwords.words('english'))
    def process_text(text):
        tokens = word_tokenize(text)
        return [word for word in tokens if word not in stop_words]

    df['tokens'] = df['headline'].apply(process_text)
    return df
```

YOUR LOGO

PART 02

嵌入技术与模型构建



嵌入技术选择

Word2Vec嵌入

Word2Vec通过上下文学习词向量，能捕捉词的语义关系。在训练数据上训练Word2Vec模型，为文本提供初始特征。

GloVe嵌入

GloVe基于全局词共现矩阵，能更好地处理词的多义性。加载预训练的GloVe向量，丰富词的语义表示。

组合嵌入矩阵

将Word2Vec和GloVe向量拼接，形成更强大的嵌入矩阵。提高模型对词义的理解能力，为深度学习模型提供更好的输入。

2.嵌入技术与模型构建：

GloVe 嵌入加载函数
这个函数的作用是读取 GloVe 预训练的词向量文件，将每个单词及其对应的向量存储到一个字典中。
每行数据格式为：单词 向量值1 向量值2 ...
函数逐行读取文件，提取单词和对应的向量，并存入字典 embeddings_index，最后返回这个字典。
组合嵌入矩阵创建函数
这个函数的作用是将 Word2Vec 和 GloVe 两种词向量结合起来，生成一个新的混合嵌入矩阵。
对于每个单词，如果它同时存在于 Word2Vec 和 GloVe 中，则将两者的向量拼接起来，存入矩阵对应位置。
最终返回一个包含所有单词混合向量的矩阵，供模型使用。
总结：
这两个函数分别用于加载 GloVe 词向量和将 GloVe 与 Word2Vec 的词向量结合起来，生成一个混合嵌入矩阵，方便后续深度学习模型使用。

```python
# GloVe嵌入加载函数
def load_glove_embeddings(glove_path):
    embeddings_index = {}
    with open(glove_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
    return embeddings_index

# 组合嵌入矩阵创建函数
def create_combined_matrix(w2v_model, glove_embeddings, word_index, embedding_dim):
    embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim * 2))
    for word, i in word_index.items():
        if word in w2v_model.wv and word in glove_embeddings:
            embedding_matrix[i] = np.concatenate(
                (w2v_model.wv[word], glove_embeddings[word])
            )
    return embedding_matrix
```

3.Keras模型构建：

模型架构设计

使用双向LSTM捕捉文本的双向依赖关系，提取时间序列特征。

加入Dropout层防止过拟合，提高模型的泛化能力。

最后通过全连接层输出分类结果，使用Sigmoid激活函数。

模型训练与优化

使用Adam优化器，动态调整学习率，加快收敛速度。

设置合适的批量大小和训练轮数，确保模型充分学习。

在训练过程中监控验证集性能，防止过拟合。

模型评估与可视化

使用混淆矩阵和分类报告评估模型性能。

可视化混淆矩阵，直观展示模型的分类效果。

分析模型在不同类别上的表现，找出不足之处。

```python
# Keras模型训练部分
def train_keras_model(df):
    print("========== Training Keras Model ==========")
    X = df['tokens']
    y = df['is_sarcastic']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Word2Vec训练
    w2v_model = Word2Vec(sentences=X_train, vector_size=100, window=5, min_count=1, workers=4)
    glove_embeddings = load_glove_embeddings(r'C:\Users\cdf\PycharmProjects\SparkRdd\kaggle\glove.twit

    # 文本序列化
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(X_train)
    X_train_seq = tokenizer.texts_to_sequences(X_train)
    X_test_seq = tokenizer.texts_to_sequences(X_test)
    X_train_pad = pad_sequences(X_train_seq, maxlen=100, padding='post')
    X_test_pad = pad_sequences(X_test_seq, maxlen=100, padding='post')
```

```
# 构建模型
model = Sequential([
    Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=200,
              input_length=100, weights=[create_combined_matrix(w2v_model, glove_embeddings, tokeni
              trainable=False),
    Bidirectional(LSTM(64, return_sequences=True)),
    Dropout(0.5),
    Bidirectional(LSTM(32)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy', optimizer=Adam(0.001), metrics=['accuracy'])
history = model.fit(X_train_pad, y_train, batch_size=64, epochs=10, validation_data=(X_test_pad, y
```

4.Spark模型构建：

Spark环境搭建

使用SparkSession创建Spark应用，处理大规模数据。

构建Spark流水线，包含分词、去除停用词和特征提取等步骤。

通过流水线简化数据处理和模型训练流程。

数据平衡与特征提取

对数据进行上采样，平衡类别分布，避免模型偏向多数类。

使用Spark的Word2Vec提取文本特征，与Keras模型进行对比。

通过流水线自动处理数据，提高模型训练效率。

模型训练与评估

使用多层感知机分类器训练Spark模型，设置合适的网络结构。

评估模型的准确率，与Keras模型进行对比。

展示预测结果示例，分析模型的输出情况。

```python
# Spark模型训练部分
def train_spark_model(df):
    print("\n========== Training Spark Model ==========")
    spark = SparkSession.builder \
        .appName("SarcasmDetection") \
        .getOrCreate()

    spark_df = spark.createDataFrame(df)

    # 平衡数据集
    majority_class = spark_df.filter(spark_df["is_sarcastic"] == 0)
    minority_class = spark_df.filter(spark_df["is_sarcastic"] == 1)
    upsampled_minority = minority_class.sample(withReplacement=True, fraction=2.0, seed=42)
    balanced_df = majority_class.union(upsampled_minority)
```

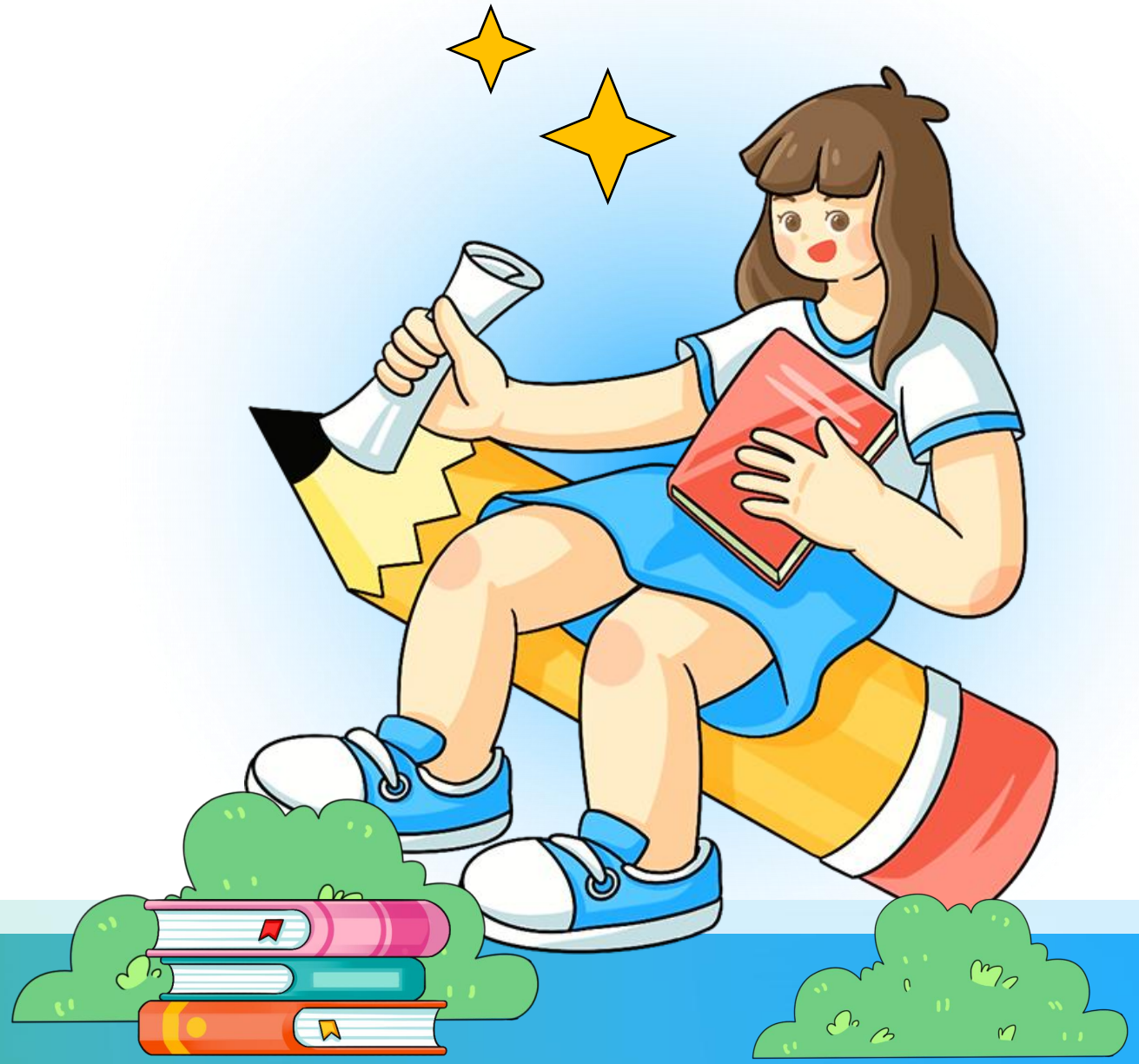
正在加载...

```
        StopWordsRemover(inputCol="words", outputCol="filtered_words"),
        SparkWord2Vec(vectorSize=200, minCount=1, windowSize=8, inputCol="filtered_words", outputCol='
        MultilayerPerceptronClassifier(
            layers=[200, 128, 64, 32, 2],
            labelCol="is_sarcastic",
            maxIter=20,
            blockSize=128,
            seed=42
        )
    ])

    # 拆分数据集并训练
    train_df, test_df = balanced_df.randomSplit([0.8, 0.2], seed=42)
    pipeline_model = pipeline.fit(train_df)
    predictions = pipeline_model.transform(test_df)

    # 模型评估
    evaluator = MulticlassClassificationEvaluator(
        labelCol="is_sarcastic",
        predictionCol="prediction",
        metricName="accuracy"
    )
    accuracy = evaluator.evaluate(predictions)
    print(f"Spark Model Test Accuracy: {accuracy:.4f}")

    # 显示预测结果示例
    print("\nSpark Model Prediction Samples:")
    predictions.select("prediction", "is_sarcastic").show(50)
    spark.stop()
# 主程序
```

YOUR LOGO

PART 03

模型对比与总结



模型性能对比

准确率对比

对比Keras模型和Spark模型在测试集上的准确率。

分析两种模型在不同数据集上的表现差异。

混淆矩阵对比

展示两种模型的混淆矩阵，对比分类效果。

分析模型在不同类别上的误分类情况。

训练效率对比

对比两种模型的训练时间，分析效率差异。

讨论模型在大规模数据上的适用性。