

摘要

Apache Spark 是用于大规模数据处理的统一分析引擎。它在存储器内运行程序的运算速度能做到比 Hadoop MapReduce 的运算速度快上 100 倍，即便是运行程序于硬盘时，Spark 也能快上 10 倍速度。此外，Spark 允许用户将数据加载至集群存储器，并多次对其进行查询，非常适合用于机器学习算法。为了更好的掌握 spark 分布式框架及集群的使用技巧，我们以高能物理研究为背景，实现机器学习算法在大规模数据集上的应用。

文本情感分析(Sentiment Analysis)是指利用自然语言处理和文本挖掘技术，对带有情感色彩的主观性文本进行分析、处理和抽取的过程。目前，情感分析研究涵盖了包括自然语言处理、文本挖掘、信息检索、信息抽取、机器学习和本体学等多个领域，得到了许多学者以及研究机构的关注，近几年持续成为自然语言处理和文本挖掘领域研究的热点问题之一。

本实验基于 Spark 大数据分布式框架，采用基于情感词典匹配和基于 TF-IDF 特征的逻辑回归的两种算法实现文本情感分析任务，以 IMDB 库的电影评论作为数据集，在分类准确率指标下衡量分类结果。

关键词：大数据，分布式计算，文本情感分析

目录

摘要.....	1
1 任务描述.....	1
2 实验数据.....	2
2.1 数据描述.....	2
2.2 数据预处理.....	2
3 实验过程及结果分析.....	4
3.1 实验环境.....	4
3.2 算法介绍.....	5
3.2.1 基于情感词典匹配的分类算法.....	5
3.2.2 基于 TF-IDF 特征的逻辑回归算法.....	5
3.3 Spark 数据处理	6
3.4 程序设计.....	7
3.4.1 基于情感词典匹配的分类算法.....	7
3.4.2 基于 TF-IDF 特征的逻辑回归算法.....	7
3.5 Spark 操作命令	8
3.5.1 集群启动.....	8
3.5.2 程序运行	8
3.5.3 节点工作状态.....	9
3.6 结果分析.....	9
3.6.1 评价指标.....	9
3.6.2 实验结果.....	9
4 实验结论.....	11

1 任务描述

文本情感分析(Sentiment Analysis)是指利用自然语言处理和文本挖掘技术，对带有情感色彩的主观性文本进行分析、处理和抽取的过程。目前，情感分析研究涵盖了包括自然语言处理、文本挖掘、信息检索、信息抽取、机器学习和本体学等多个领域，得到了许多学者以及研究机构的关注，近几年持续成为自然语言处理和文本挖掘领域研究的热点问题之一。

文本情感分析的一个基本步骤是对文本中的某段已知文字的两极性进行分类，这个分类可能是在句子级、功能级。分类的作用就是判断出此文字中表述的观点是积极的、消极的、还是中性的情绪。更高级的“超出两极性”的情感分析还会寻找更复杂的情绪状态，比如“生气”、“悲伤”、“快乐”等等。有很多开源软件使用机器学习、统计、自然语言处理的技术来计算大型文本集的情感分析，这些大型文本集合包括网页、网络新闻、网上讨论群、网络评论、博客和社交媒介。

本实验基于 Spark 大数据分布式框架，采用基于情感词典匹配和基于 TF-IDF 特征的逻辑回归的两种算法实现文本情感分析任务，以 IMDB 库的电影评论作为数据集，在分类准确率指标下衡量分类结果。

2 实验数据

2.1 数据描述

本次实验选用的源数据来自 IMDB 库的电影评论数据集（地址：<https://www.kaggle.com/datasets/ebiswas/imdb-review-dataset>）。IMDB 是全球最大的电影、电视节目和流媒体数据库之一，源数据集一共包含网站用户群体针对 453,528 部作品的 5,571,499 条评论数据，实验中我们选取了其中 35,000 条评论进行计算。

我们的数据集中每条数据包含 3 个字段‘id’、‘review’和‘sentiment’，分别对应评论编号、评论内容和评论情感（如表 2-1），其中评论内容均为英文文本。

表 2-1 数据集中每条数据包含的字段

属性字段	注释
id	针对每条评论的唯一编号
review	评论内容
sentiment	1=积极情感 & 0=消极情感

2.2 数据预处理

由于源数据集中的每条数据包含 9 个属性字段（如表 2-1），且评分划分较细，而我们实验设计的是一个文本情感二分类任务，因此我们对源数据进行了预处理。我们提取出其中的‘review_id’、‘rating’和‘review_detail’3 个字段的内容，并只考虑评分的两级分化，将评分 ≤ 5 分的评论视为负面评论，评分 ≥ 6 分的评论视为正面评论。数据预处理代码见图 2-1。

表 2-2 源数据集中每条数据包含的字段

属性字段	注释
review_id	针对每条评论的唯一编号
reviewer	评论者的用户名
movie	影片名
rating	评分（0-10）
review_summary	评论的基本摘要
review_date	评论发表的时间
spoiler_tag	1=剧透 & 0=不剧透
review_detail	评论内容
helpful	1=有用 & 0=无用

```

1 import json
2 import pandas as pd
3
4 with open('archive/part-01.json', 'r') as f:
5     part1 = json.load(f)
6
7 id = []
8 review = []
9 score = []
10
11 for data in part1:
12     id.append(data['review_id'])
13     review.append(data['review_detail'])
14     if data['rating'] in ['6', '7', '8', '9', '10']:
15         score.append('1')
16     else:
17         score.append('0')
18
19
20 dataset = list(zip(id, review, score))
21 df = pd.DataFrame(data=dataset, columns=['id', 'review', 'score'])
22 df.to_csv('IMDB1.csv', index=False, header=True)

```

图 2-1 数据预处理代码

3 实验过程及结果分析

3.1 实验环境

MapReduce 是 Google 提出的一种并行编程框架，用于简化分布式编程和处理大规模数据。Hadoop 是 MapReduce 的开源实现。Hadoop 本身的计算模型决定了 Hadoop 上的所有工作都必须转化为 Map、Shuffle、Reduce 等核心阶段。由于每次都必须从磁盘读取或写入数据，而整个计算模型都需要网络传输，这就导致了越来越难以忍受的延迟。同时，任何任务都必须等上一个任务完成后才能运行，直接导致其无法支持交互应用。Apache Spark 是一个开源集群运算框架，最初是由加州大学柏克莱分校 AMPLab 所开发。相对于 Hadoop 的 MapReduce 会在执行完工作后将中介资料存放到磁盘中，Spark 使用了存储器内运算技术，能在资料尚未写入硬盘时即在存储器内分析运算。Spark 在存储器内执行程序的运算速度能做到比 Hadoop MapReduce 的运算速度快上 100 倍，即便是执行程序于硬盘时，Spark 也能快上 10 倍速度。Spark 允许用户将资料加载至集群存储器，并多次对其进行查询，非常适合用于机器学习算法。

使用 Spark 需要搭配集群管理员和分布式存储系统。Spark 支持独立模式（本地 Spark 集群）、Hadoop YARN 或 Apache Mesos 的集群管理。在分布式存储方面，Spark 可以和 Alluxio、HDFS、Cassandra、OpenStack Swift 和 Amazon S3 等接口搭配。

实验中我们将三台台式机通过同一交换机连接到同一个局域网内，使得它们之间可以彼此通信。三台机器中一台机器作为 master 节点，同时三台机器都作为 worker 节点。我们通过 HDFS 系统进行文件的管理，通过编写 Python 程序然后通过 submit 命令提交 spark 执行，使得任务可以在分布式环境上运行。物理环境的配置如表 3-1、表 3-2 所示。

表 3-1 节点硬件配置表

节点	IP 地址	内存	Cores
Master	172.24.71.22	8G	4
Worker1	172.24.64.195	8G	4
Worker2	172.24.71.22	8G	4

表 3-2 节点环境配置表

系统版本	Ubuntu 20.04
JDK 版本	1.8.0_41
Hadoop 版本	3.3.1

3.2 算法介绍

在本次实验中，我们选用两种算法对文本情感进行预测，第一种为基于情感词典匹配的分类算法，第二种为基于 TF-IDF 特征的逻辑回归算法。TF-IDF 是一种用于信息检索与数据挖掘的常用加权技术，常用于挖掘文章中的关键词，其优点是计算简单快速，而且容易理解；而逻辑回归模型(Logistic Regression, LR)，又称对数几率模型，是机器学习中的一种分类模型，算法的简单和高效，在实际应用中被广泛应用。

3.2.1 基于情感词典匹配的分类算法

在基于情感词典匹配的分类算法中，我们用到了一组预训练的、标记好情感类别的词典，通过将分词结果和词典中的正/负向词汇进行比对，从而计算出句子中所含正/负向词汇的比例来推测文本的情感倾向。例如，在预训练的词典中，'adore'、'enjoy'被标记为正向情感词汇，那么包含这些词汇比例高，而其他负向情感词汇比例低的文档就很容易被判定为消极文本。

3.2.2 基于 TF-IDF 特征的逻辑回归算法

TF-IDF 中的 TF 和 IDF 分别代表词频和逆文档频率，其计算公式为：

$$\text{词频}(TF) = \frac{\text{某个词在文章中出现次数}}{\text{文章的总次数}} \quad (3-1)$$

$$\text{逆文档频率}(IDF) = \log \frac{\text{语料库的文档总数}}{\text{包含该词的文档数} + 1} \quad (3-2)$$

$$TF - IDF = TF \times IDF \quad (3-3)$$

可以看出，某个词在文章中的 TF-IDF 越大，那么一般而言这个词在这篇文章的重要性会越高，所以通过计算文章中各个词的 TF-IDF，由大到小排序，排在最前面的几个词，就是该文章的关键词。

如果忽略二分类问题中Y的取值是一个离散的取值（0 或 1），使用线性回归来预测Y的取值。这样做会导致Y的取值并不为 0 或 1。逻辑回归使用一个函数来归一化Y值，使Y的取值在区间(0,1)内，这个函数称为 Logistic 函数(Logistic function)，也称为 Sigmoid 函数(sigmoid function)，公式如下：

$$g(z) = \frac{1}{1 + e^{-z}} \quad (3-4)$$

二项逻辑回归模型是一种二分类模型，由条件概率分布 $P(Y|X)$ 表示，形式为参数化的 Logistic 分布。这里，随机变量 X 为实数，随机变量 Y 取值为 0 或 1，其条件概率分布可以写作：

$$P(Y = 1|x) = \frac{\exp(w \cdot x + b)}{1 + \exp(w \cdot x + b)} \quad (3-5)$$

$$P(Y = 0|x) = \frac{1}{1 + \exp(w \cdot x + b)} \quad (3-6)$$

其中， $x \in \mathbb{R}^n$ 为输入， $x \in \{0,1\}$ 为分类结果， $w \in \mathbb{R}^n$ 和 $b \in \mathbb{R}$ 为模型参数。

3.3 Spark 数据处理

本实验主要使用 pyspark 的 DataFrame 和 RDD 数据格式进行处理；将对数据清洗的代码整合到 clean 函数中，然后将 DataFrame 格式转化为 RDD 格式，使用 pyspark 的并行化函数调用 clean 进行快速处理。代码如图 3-1。

```
# 下面是使用pyspark的写法
from pyspark.sql import Row
def clean(item):
    review = item['review']
    s = BeautifulSoup(review, 'lxml')
    s_soup = s.get_text()
    strip = re.sub(comb, '', s_soup)
    letter = re.sub("[^a-zA-Z]", " ", strip)
    lower = letter.lower()
    word = tok.tokenize(lower)
    res = (" ".join(word)).strip()
    # 由于item是PySpark的Row对象，与tuple类似，无法进行修改，所以创建一个新的Row对象
    ans = Row(id=item['id'],sentiment=item['score'], review=res)
    # item['review'] = res # 无法进行修改
    return ans

# new_df = df.select("review").collect() # 用SQL语句进行选择，返回是list
# print(type(new_df))
# 并行化，调用处理函数
df.dropna(how='any')
new_df = spark.sparkContext.parallelize(df.collect()).map(clean) # 类型是RDD.PipelineRDD
print(type(new_df))
```

图 3-1 Spark 数据处理代码

将处理后的评论数据整合回原来的 DataFrame 格式，然后存储到 HDFS 特定文件夹下。代码如图 3-2。


```

# 将RDD数据转化为DataFrame, 需要定义列和类型
from pyspark.sql.types import ArrayType, StructField, StructType, StringType, IntegerType
schema = StructType([
    StructField('id', StringType(), True),
    StructField('sentiment', IntegerType(), True),
    StructField('review', StringType(), True)
])
# 转换为csv
# pysparkDF = new_df.toDF() # 缺乏类型
pysparkDF = spark.createDataFrame(new_df, schema)
pysparkDF.dropna(how='any')
print(type(pysparkDF))

# pysparkDF = spark.createDataFrame(new_df)

pysparkDF.write.mode('overwrite').format("csv").option("encoding", "utf-8").option("header", True).save("/test/imdb_clean_path")

```

图 3-2 处理后的数据存储代码

最终，我们得到处理后大小为 251.9M 的数据。数据量大小统计如图 3-3。

```

hadoop@hdr:/usr/spark-3.2.1$ hdfs dfs -count -h /test/imdb_clean_path
1              4              251.9 M /test/imdb_clean_path

```

图 3-3 数据量大小统计

3.4 程序设计

3.4.1 基于情感词典匹配的分类算法

首先读取 HDFS 文件夹下的情感词典，然后使用 pyspark 的 select 语句选出电影评论，然后与情感词典做交集运算。每个评论与情感词典交集词汇的得分总和，便是该评论的情感得分，然后据此进行分类。代码如图 3-4。

```

#reading the file in a dataframe.
bucket_df = sqlContext.read.format('com.databricks.spark.csv').options(header = 'true', inferschema='true').load('hdfs://172.24.71.22:9000/test/sent')
#creating a tokenizer object for splitting the input review column and creating a new output column 'split'
token = RegexTokenizer().setGaps(False).setPattern("\\p{L}+").setInputCol("review").setOutputCol("split") #
#used the token transformer for applying the changes.
df_word = token.transform(imdb_clean_df)
# print(df_word.show(6))
#We will compare with the bucket words by exploding the split words into different rows using f.explode function and join the dataframes.
joined_df = df_word.select('id', fn.explode('split').alias('word')).join(bucket_df, 'word')
predict_df = joined_df.groupBy('id').agg(fn.avg('sentiment').alias('avg_sent')).withColumn('pred_sent', fn.when(fn.col('avg_sent') > 0, 1).otherwise
print(imdb_clean_df.join(predict_df, 'id').select(fn.expr('float(sentiment = pred_sent)').alias('accuracy')).select(fn.avg('accuracy')).show(5))

```

图 3-4 基于情感词典匹配的算法实现

3.4.2 基于 TF-IDF 特征的逻辑回归算法

在统计词频时，去除调停用词，减少其对实验结果的干扰；然后构造 TF-IDF 特征。这里我们使用了 pyspark ML 提供的高级 API，这种高级 API 构建在 DataFrame 之上，能够将多种算法合并成一个 pipeline。

```

#removing the stop words.
englishStopWords = StopWordsRemover.loadDefaultStopWords("english")
filter = StopWordsRemover().setStopWords(englishStopWords).setCaseSensitive(False).setInputCol("split").setOutputCol("filter_words")

# we will remove the words that appear in 8 docs or less.
count_vector = CountVectorizer(minTF=1, minDF=8, vocabSize=2**17).setInputCol("filter_words").setOutputCol("TF")

# we now create a pipelined estimator.
CVP = Pipeline(stages=[token, filter, count_vector]).fit(imdb_clean_df)

#applying the transformation to the dataframe.
CVP.transform(imdb_clean_df)

IDF = IDF().setInputCol('TF').setOutputCol('TF_IDF')
#creating an IDF pipeline
IDF_pip = Pipeline(stages=[CVP, IDF]).fit(imdb_clean_df)#applying the transformation.
TF_IDF = IDF_pip.transform(imdb_clean_df)
TF_IDF.select('id', 'filter_words', 'TF', "TF_IDF").show(4)

train_df, val_df, test_df = imdb_clean_df.randomSplit([0.80, 0.1, 0.1], seed=0)
print(train_df.count(), val_df.count(), test_df.count())

# Logistic Regression Model
#using the TF_IDF features from IDF pipeline and feeding it to the Logistic regression model.
LR = LogisticRegression().setLabelCol('sentiment').setFeaturesCol('TF_IDF').setMaxIter(100)
#creating a pipeline transformation for logistic regression and running on the training data set.
LR_pip = Pipeline(stages=[IDF_pip, LR]).fit(train_df)#applying the transformation on the validation data frame for predicting the sentiments.
LR_pip.transform(val_df).select('id', 'sentiment', 'prediction').show(5)
#applying the transformation on the test data frame
pred_test = LR_pip.transform(test_df).select('id', 'sentiment', 'prediction')
# pred_test.show(5)

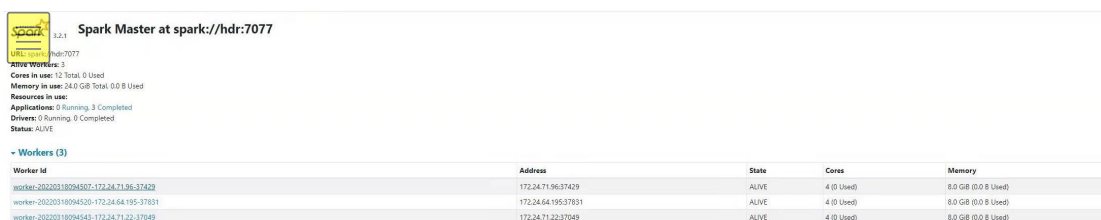
```

图 3-5 基于 TF-IDF 特征的逻辑回归算法实现

3.5 Spark 操作命令

3.5.1 集群启动

根据现有的环境配置，在 hadoop 的 sbin 目录下输入: \$./start all.sh 即可启动 Hadoop 集群。之后再 Spark 的 sbin 目录下输入 \$ stasrt master.sh 和 \$ start slaves.sh 即可启动 Spark 集群。启动成功后可在 172.24.71.22: 8080 界面看到如图 3-6 信息。



Worker Id	Address	State	Cores	Memory
worker-20220318094507-172.24.71.96-37429	172.24.71.96:37429	ALIVE	4 (0 Used)	8.0 GB (0.0 B Used)
worker-20220318094520-172.24.64.195-37831	172.24.64.195:37831	ALIVE	4 (0 Used)	8.0 GB (0.0 B Used)
worker-20220318094543-172.24.71.22-37049	172.24.71.22:37049	ALIVE	4 (0 Used)	8.0 GB (0.0 B Used)

图 3-6 Spark 节点信息

3.5.2 程序运行

进入 spark 安装目录，输入 bin/spark-submit --master spark://172.24.71.22:7077 /usr/hadoop-3.3.1/sparkdata/sa_final.py --driver-memory 8G --excutor-memory 8G --executor-cores 4，即可利用 Spark 集群进行程序运行。

3.5.3 节点工作状态

图 3-7 为程序分布式运行时各节点的工作状态，其中，两个子节点被随机分配任务，可能出现某个子节点空闲的情况。

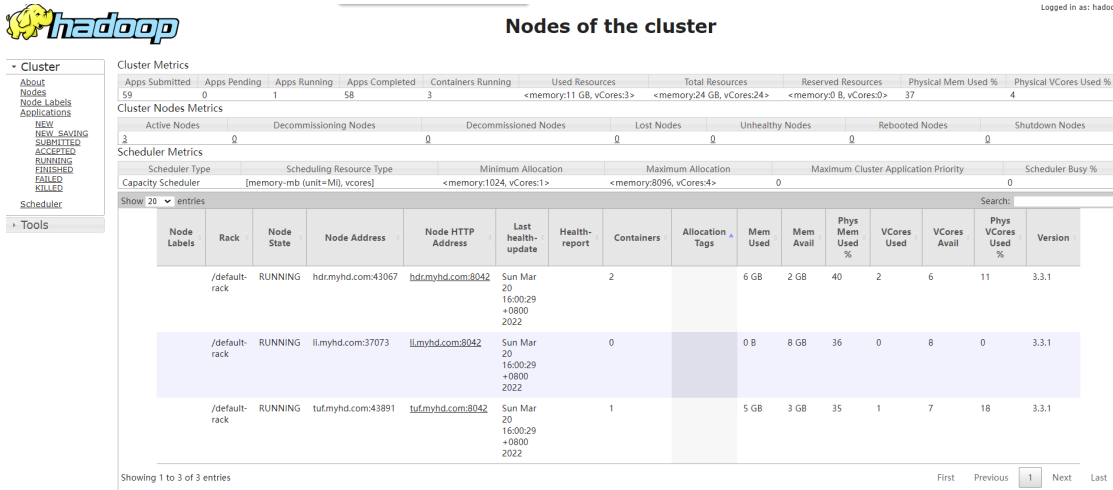


图 3-7 节点工作状态查看

3.6 结果分析

3.6.1 评价指标

实验选用的评价指标为 $accuracy_score$ （分类准确率分数），即所有分类正确的百分比。计算公式为：

$$accuracy_score = \frac{TP + TN}{TP + TN + FP + FN} \quad (3-7)$$

3.6.2 实验结果

程序运行结果如图 3-8。可以看到，实验中基于情感词典匹配的分类算法在我们的数据集下计算的分类准确率为 55.997%，基于 TF-IDF 特征的逻辑回归算法的准确率为 53.616%。

```

+-----+-----+-----+
|      id|sentiment|prediction|
+-----+-----+-----+
|rw5710685|      0.0|      0.0|
|rw6053283|      1.0|      0.0|
|rw2479780|      0.0|      1.0|
|rw6056695|      0.0|      1.0|
|rw5154969|      0.0|      0.0|
+-----+-----+-----+
only showing top 5 rows

```

(a)

```

+-----+
|      avg(accuracy)|
+-----+
|0.559972000861512|
+-----+

```

(b)

```

+-----+-----+-----+-----+
|      id|      filter_words|      TF|      TF_IDF|
+-----+-----+-----+-----+
|rw5704482|[enjoyed, first, ...|(34138,[2,3,9,10,...|(34138,[2,3,9,10,...|
|rw5704483|[know, iceland, s...|(34138,[15,35,67,...|(34138,[15,35,67,...|
|rw5704484|[truth, much, mov...|(34138,[0,1,3,4,5...|(34138,[0,1,3,4,5...|
|rw5704485|[seen, film, firs...|(34138,[0,1,2,7,8...|(34138,[0,1,2,7,8...|
+-----+-----+-----+-----+
only showing top 4 rows

```

(c)

```

2022-03-20 16:03:49,372 WARN scheduler.DAGScheduler: Broadcasting large task binary with size 1221.5 KiB
0.5361598539405619

```

(d)

图 3-8 (a) 基于情感词典匹配的分类算法的判别结果和标签值的对比（仅显示前 5 行）；(b) 基于情感词典匹配的分类算法的分类准确率；(c) 基于 TF-IDF 特征的 1 逻辑回归算法提取出的对应文本的 TF-IDF 特征；(d) 基于 TF-IDF 特征的逻辑回归算法的分类准确率。

结果分析：基于情感词典匹配的算法的结果并不是很理想的原因来自我们采用的情感词典是基于另外一个领域的数据集上，不能很好地覆盖到电影评价的情感词；基于 TD-IDF 特征的逻辑回归算法采用的特征比较简单，并且由于计算资源有限，我们设计的训练周期也偏少。

4 实验结论

本实验通过基于情感词典匹配、和基于 TF-IDF 特征的逻辑回归算法对 IMDB 电影评价数据集进行了训练和测试。基于情感词典匹配的算法最后准确率为 0.55，而基于 TF-IDF 特征的逻辑回归算法的最后准确率为 0.53。可能的原因是 TF-IDF 特征并不能很好地对句子的语义建模，所以效果并不如使用专家定义的情感词典分析更有效。这也提示我们在面临语义相关的自然语言处理任务，简单的词频特征并不能带来好的效果。

通过本次实验，我们对 Spark 分布式框架及集群的使用有了更深层次的理解与感悟，将对我们以后的学术研究以及就业工作有很大帮助。