中文垃圾邮件分类

1.数据预处理

- 首先导入数据，创建dataframe

```
## train set
pandas_train_df = pd.read_csv('./train.csv')
pyspark_train_df = spark.createDataFrame(pandas_train_df)
## test set
pandas_test_df = pd.read_csv('./test.csv')
pyspark_test_df = spark.createDataFrame(pandas_test_df)

+--------------------+-----+
|             content|label|
+--------------------+-----+
|Received: from to...|    1|
|Received: from gr...|    0|
|Received: from 16...|    1|
|Received: from go...|    1|
|Received: from se...|    0|
|Received: from so...|    0|
```

- 对文本进行预处理，包括提取中文，分词，去除停用词。与pandas的apply操作类似，pyspark为我们提供了udf方法，可以对每一行元素施加我们自己定义的函数。并用withColumn函数添加新的列。

```
def pre_proc(content):
    pattern = "[\u4e00-\u9fa5]+"
    regex = re.compile(pattern)
    mail_content = ''.join(regex.findall(content))
    cut_mail_content = (' '.join([k for k in jieba.lcut(mail_content, cut_all=True) if k not in stopwords]))
    return cut_mail_content

pre_proc_udf = udf(pre_proc,StringType())
pyspark_train_df = pyspark_train_df.withColumn("content",pre_proc_udf(pyspark_train_df["content"]))
pyspark_test_df = pyspark_test_df.withColumn("content",pre_proc_udf(pyspark_test_df["content"]))

+--------------------------------+-----+
|                         content|label|
+--------------------------------+-----+
|特价 订 全国 酒店 酒店客房 店...|    1|
|标题 真 郁闷 烦死 烦死人 什么...|    0|
|网站 首页 游戏 首页 小游戏 游...|    1|
|                                |    1|
|没有 可能 当断则断 断去 去年 ...|    0|
|以下 下文 文字 转载 转载自 自...|    0|
```

```
21 |贵 公司 负责 负责人 责人 经理...|    1|
```

2.文本的数字表示

这里我们采用的是tf-idf对文本进行编码。具体来说

$$词频（TF） = \frac{某个词在文章中出现的次数}{文章的总词数}$$

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

$$TF - IDF = 词频（TF） * 逆文档频率（IDF）$$

该编码方式的优缺点

- 优点：简单快速，容易理解
- 缺点：用词频衡量一个词的重要程度不够全面。但在本任务中，垃圾邮件的标志往往都是：“贵”，“公司”，“顾客”这些高频词，故可以使用该编码方式

pyspark中对于tf-idf的计算已经封装好了，其中tokenizer是为了将字符串转为列表，传给hashingtf进行计算。下面是第一封邮件转化后的表示。

```
tokenizer =Tokenizer(inputCol="content", outputCol="words")
hashingTF = HashingTF(inputCol = "words",outputCol = "rawfeatures")
idf = IDF(inputCol = "rawfeatures",outputCol = "features")


Row(features=SparseVector(262144, {6618: 3.9619, 8271: 6.2556, 9592: 2.9629,
10772: 2.9266, 18249: 4.8567, 20378: 7.7932, 21641: 5.7533, 22682: 3.8731,
22878: 2.53, 26405: 4.3995, 26408: 22.5096, 27670: 3.071, 30792: 3.7681,
32371: 3.119, 37376: 7.1755, 38117: 2.4244, 43487: 5.2474, 44305: 4.0115,
44996: 4.2506, 47830: 6.4837, 52353: 3.0617, 54961: 5.308, 55015: 5.3277,
56436: 5.5253, 59600: 3.6139, 60056: 4.8517, 60848: 4.0463, 66390: 3.5954,
66775: 5.0601, 71691: 2.2505, 74912: 17.6709, 78338: 2.2262, 80092: 2.0854,
83373: 2.9693, 84597: 3.3211, 86921: 4.6187, 87022: 5.2328, 104751: 4.3194,
108143: 4.1849, 108657: 1.819, 110966: 2.3019, 111283: 4.3107, 111442:
17.2055, 112769: 5.6658, 113680: 5.1798, 117267: 4.6187, 120894: 5.3936,
121917: 2.1713, 122084: 4.8105, 124597: 1.6639, 125920: 5.4595, 129561:
4.4027, 133896: 5.5301, 134359: 1.6247, 135047: 11.3315, 137336: 2.6726,
138762: 4.0835, 139369: 12.2163, 141610: 4.5672, 142852: 3.1656, 143368:
75.5586, 143849: 26.386, 144476: 4.0893, 144714: 10.7112, 148833: 8.4109,
152157: 5.2007, 164820: 5.6474, 166608: 5.4371, 172608: 5.6382, 183052:
8.4314, 183800: 9.2375, 186565: 4.4713, 187699: 6.1982, 189216: 4.8081,
189483: 11.2983, 193914: 3.4272, 196480: 3.7457, 196489: 3.74, 215158:
9.6838, 216039: 5.5301, 216859: 9.0293, 218718: 2.325, 222878: 9.2612,
224073: 2.9034, 228269: 4.6385, 228718: 4.6628, 234431: 5.3002, 234589:
5.1263, 237312: 1.0238, 237879: 4.5058, 239624: 5.4501, 244815: 5.5649,
246573: 4.9754, 250406: 5.6437, 258277: 46.6837, 258488: 3.68}))
```

3.模型选择

我们在尝试了多个模型（朴素贝叶斯，逻辑回归，决策树）后，选择了表现最好的一个：逻辑回归

逻辑回归对于模型做出如下假设，其中的数学推导在此就不赘述了

$$P(y=1|x;\theta)=g(\theta^T x)=\frac{1}{1+e^{-\theta^T x}}$$

同样的，pyspark为我们封装好了函数供我们使用

```
lr = LogisticRegression(maxIter = max_iter)
```

连同上一步的3个操作，我们可以把这四个操作集成为一个pipeline

```
pipeline = Pipeline(stages = [tokenizer,hashingTF,idf,lr])
```

PipeLine是基于DataFrames的高层的API，可以方便用户构建和调试机器学习流水线.

4.模型训练与预测

构建好了pipeline，训练模型只需要把训练集fit到pipeline中

```
model = pipeline.fit(pyspark_train_df)
```

预测时只需要对测试集进行transform

```
prediction = model.transform(pyspark_test_df).select("prediction").toPandas()
```

最后再将结果输出到txt文件中

```
f = open('result.txt','w')
for i in range(len(prediction["prediction"])):
    f.write(str(int(prediction["prediction"][i]))+'\n')
f.close()
```

从结果上看，该模型的准确率是很高的，在测试集上能够达到99.14%的准确率

5.小组分工

没有具体的分工，所有工作都是共同商讨完成的。虽然代码量不大，但是每一步都需要学习spark自带的操作，并且最终的代码只包含最好的模型。

完整代码见pyspark_final.py文件

标注结果见new_result.txt文件