网络数据挖掘大作业

分类算法实现及实验报告

# 1 引文

本次大作业的选题为垃圾短信分类。在老师所给的垃圾短信数据集上，分别实现了逻辑回归、决策树、SVM、GBDT这个四个分类算法。这四个算法都是老师上课提到的算法，通过动手实验，并进行对比，衡量了算法之间的优劣性，并给出了在不同情况下的选择理由。

搭建了Hadoop和Spark分布式环境，并使用Spark MLlib实现了分布式的逻辑回归算法。通过测试，对比了分布式实现机器学习算法和单机的区别。并分析了如何设计一个高效并节省空间的分布式机器学习算法。

最后设计并实现了一个可视化Web系统，展示了对老师所给垃圾短信数据集的分类结果，以及不同算法的正确率等指标对比。系统在47.104.152.193:8000/index/上部署，并在提交作用中附有使用说明文档。

# 2 相关工作

虽然我们为了巩固课堂老师所讲解的分类算法知识，只实现了老师课堂涉及的分类算法。如逻辑回归、决策树、GBDT等。但是我们课下依旧学习并了解分类算法的研究进展以及相关工作。

文本分类算是自然语言处理一个相当经典的问题。传统的文本分类方法分为特征工程和分类器两个部分，在特征工程中首先对文本进行预处理，传统算法主要有基于字符串匹配的正向/逆向/双向最大匹配；基于理解的句法和语义分析消歧等。近年来随着深度学习的应用，WordEmbedding + Bi-LSTM+CRF方法逐渐成为主流

然后从处理后的数据中提取特征，并用常用词袋模型或向量空间模型等进行文本表示，或者传统做法还有LDA主题模型等进行基于语义的表示。最后接一个分类器，例如逻辑回归，SVM等算法进行分类。现在流行的深度学习文本分类方法，主要是通过word2vec解决了文本的表示不适合深度学习的问题，在这个基础上，涌现出了很多表现不错的文本方法如TextCNN、TextRNN、TextRNN+Attention、TextRCNN等。

# 3 逻辑回归算法实现

## 3.1文本处理

### 3.1.1分词

采用结巴分词的精确模式来对短信文本进行分词操作。

### 3.1.2去停用词

停用词是一些完全没有用或者没有意义的词，例如助词、语气词等，其在大量文本中以近似的概率出现，不能提现文本特征，所以应该进行去除。

本实验选取了哈工大停用词表，共计767个停用词。

## 3.2特征提取

采用TF-iDF。由于文本是非结构化数据，长度不一的字符序列是无法直接作为算法的输入。为了解决这个问题，可以用空间模型来描述文本向量。scikit-learn提供了几个常用的文本特征提取的方法，本实验采用了TF-IDF的方法。TF-IDF是一种加权技术，采用一种统计方法，根据字词在文本中出现的次数和在整个语料中出现的文档频率来计算一个字词在整个语料中的重要程度。其中，词频（TF）为某个词在文章中出现的总次数/文章的总词数，逆文档频率（IDF）为log（词料库的文档总数/包含该词的文档数+1）。

TF-IDF的优点是能过滤掉一些常见的却无关紧要的词语，同时保留影响整个文本的重要词语。缺点是，TF-IDF的简单结构并不能有效地反映单词的重要程度和特征词的分布情况，使其无法很好的完成对权值的调整功能，所以在一定程度上该算法的精度并不是很高。

本实验采用了scikit-learn提供的TfidfTransformer和CountVectorizer方法来进行特征的提取处理。

## 3.3 LR模型

### 3.3.1模型简介

逻辑回归是一种有监督的统计学习方法，主要应用于样本的分类问题，逻辑回归分类器是以伯努利分布为模型建模，用于区分两种类别。逻辑回归模型与线性回归模型相比，就是在其基础上使用了一个假设函数，该假设函数如下所示：



从整体上来说，逻辑回归模型的使用，可以将整个实数范围上的某一值x映射到有限范围的某一点上，从而实现对某一值x的分类，也就是说对于某一样本x，通过逻辑回归分析后，可以将其归入某一类别y中。正是由于逻辑回归模型具有较好的泛化性和可解释性，使得它成为机器学习中一个重要的分类模型。

## 3.4 实验结果

本次实验，利用提供的80万带标签的短信，对其进行文本处理和特征提取，取其中80%为训练数据，其中20%为测试数据。

表3-1 RESULT(l1正则项)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | support |
| 0 | 0.99074 | 0.99781 | 0.99426 | 144143 |
| 1 | 0.97869 | 0.91518 | 0.94587 | 15857 |
| avg/total | 0.98954 | 0.98962 | 0.98946 | 160000 |

表3-2 RESULT(l2正则项)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | support |
| 0 | 0.99224 | 0.99732 | 0.99426 | 144143 |
| 1 | 0.97440 | 0.92912 | 0.95122 | 15857 |
| avg/total | 0.99047 | 0.99056 | 0.99046 | 160000 |

模型训练耗时5.532684803009033s

分词处理耗时142.29429030418396s

Tf-idf处理耗时 23.64095425605774s

## 3.5 优缺点分析

优点：形式简单，易于实现和理解，计算代价不高，模型效果不错且训练速度快，资源占用小。

缺点：准确率不是特别高，且容易欠拟合，很难处理数据不平衡的问题，对数据特征有缺失或者特征空间很大时表现效果不好，LR本身无法筛选特征且无法处理非线性数据。

# 4 GBDT算法实现

## 4.1算法思想

Boosting的基本思想就是根据初始训练数据训练出一个基学习器，加大基学习器分错的样本的比重来调整样本，用调整后的样本，训练下一个基学习器，重复T次后，将T个学习器加权结合，得到最后训练的模型。而Gradient Boosting作为boosting的其中一种方法，它每次训练新的基学习器，是使上一个模型的损失函数下降的方向调整得来的。当GB中每个学习器使用的是决策树时，这个算法就被成为GBDT，也就是我们使用的GBDT算法。

## 4.2实现

短信文本的分类属于自然语言文本处理，分成以下四步进行处理。

### 4.2.1分词及去停用词

选用了第三方库jieba（https://github.com/fxsjy/jieba）进行中文分词。Jieba分词包含了精准模式、搜索引擎模式以及全模式，对于垃圾文本分类，精准模型最契合需求，而且基于jieba分词自带的词库就可以很好的分割短信文本。

由于中文文本中包含“的”“了”等对于分类没有意义的词，因此，需要去除这些词。从网络上下载了停用词表，将分词后的每一条短信文本中包含在停用词表中的词去除。

### 4.2.2提取特征

选用TF-IDF方法来计算特征向量。通过引入第三方库scikit-learn中的CountVectorizer模型计算词频，再使用TfidfTransformer模型根据词频计算TF-IDF值。将分词并去除停用词后的短信文本列表依次输入到两个模型中计算出特征向量。

需要注意的是，因为进行线上测试时，对于需要预测的短信文本同样需要计算特征向量，它们的特征向量应该和训练文本的特征向量的维度、特征保持一致，每个单词在所有的文本中出现的次数（即IDF值）也应保持一致，而这两个模型在fit后分别生成了向量特征和IDF值，因此在fit\_transform后，保存CountVectorizer和TfidfTransformer这两个模型即可。这里使用了python自带的pickle保存模型，将这两个模型保存在了”../model\_files/”文件夹下，方便预测时使用。

### 4.2.3降维

由于提取的特征向量维度高达36万多，不进行降维时训练模型速度很慢，因此对提取的特征向量进行降维。开始选用scikit-learn中的PCA方法进行降维，出现了Memery Error。查阅资料发现，上一步中提取出来的特征向量是一个极其稀疏的矩阵，是按照scipy库中的sparse矩阵存储的，只存储了非零的值，而PCA降维需要先将这个矩阵展开成稠密矩阵后再运算，从而导致了内存不足的问题。sklearn库中的TruncatedSVD类可以解决这个问题，它直接在稀疏矩阵上展开奇异值分解，实现降维，因此最后选用这个方法降维。开始直接使用的TruncatedSVD中默认的“randomized”算法，并使用fit\_transform()一步实现降维，仍然出现了内存不够的情况。官方文档中说明“arpack”算法更加适用于稀疏矩阵的降维。经过调整TruncatedSVD模型的分类算法，以及将训练和转换分成两部实现，即依次调用fit()和transform()方法解决了内存不够的问题。同样的，与特征提取模型一样，降维模型也需要保存下来以供预测时使用。这里使用了pickle将模型保存到“../model\_files/”路径下。

### 4.2.4训练模型

开始选用了scikit-learn中GBDT类，发现计算耗时非常长。查阅资料发现，微软发布了一个开源包-----lightGBM高效地实现了GBDT算法。将降维后的特征向量及相应的标记输入到模型中训练，即可得到训练好的分类器，使用sklearn.externals.joblib保存到文件中，预测时可直接加载模型使用。

## 4.3实验

### 4.3.1 sklearn.GBDT与lightgbm.lgbm对比

使用全部的80万条标记数据训练，5折交叉验证；两个算法中的弱分类器的数目，即参数n\_estimaters，都设为100；降维后向量的维度设为100。

两个算法的实验结果如下图：

（1）GBDT的实验结果：

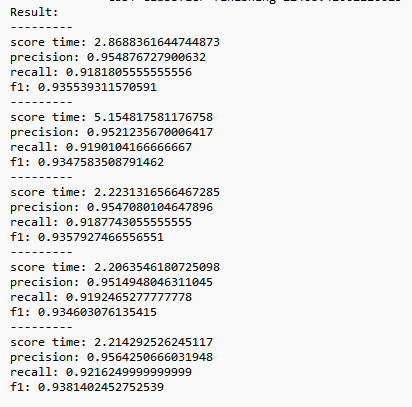


图4-1 GBDT的实验结果

（2）LGBM的实验结果：

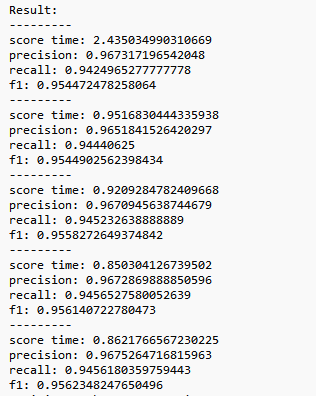


图4-2 LGBM的实验结果

可以清楚的看到，LGBM比GBDT的准确率、召回率高1%-2%左右。同时，GBDT的模型训练时长为22468.41062116623s，而LGBM的模型训练时长仅为302.307497s，将近是GBDT的运行时间的1/80。因此，最终选用LGBM作为boosting部分的模型。

### 4.3.2 确定降维维度

使用全部的80万条标记数据，10%作为测试数据；弱分类器的数目，即参数n\_estimaters，都设为1000；分别测试降维后向量的维度设为100、200、300、400。实验结果如下图：

当n\_estimaters=100时，结果如图4-3所示。

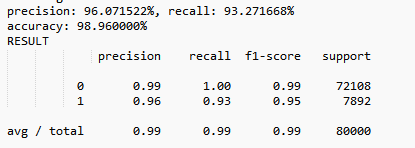


图4-3 n\_estimaters 100

当n\_estimaters=200时，结果如图4-4所示。

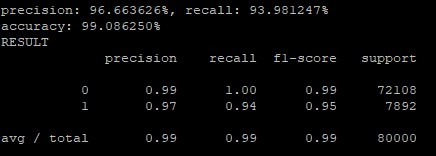


图4-4 n\_estimaters 200

当n\_estimaters=300时，结果如图4-5所示。

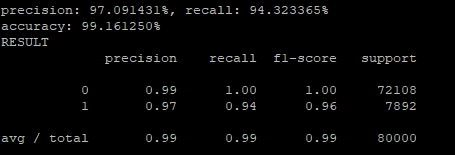


图4-5 n\_estimaters 300

当n\_estimaters=400时，结果如图4-6所示。

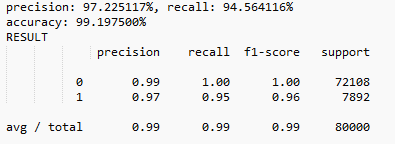


图4-6 n\_estimaters 400

可以看到，前三个实验的结果，模型的准确率提升比较明显，大概在0.1%左右，当降维维度增加到400维时，模型的分数提升已经不是很明显了。因此，选择300维作为降维后的特征向量的维数。

### 4.3.3 确定弱分类器数

LGBM中一个非常重要的参数是弱分类器的个数，即参数n\_estimaters。lgbm类中提供了一个cv方法可以找到最优模型。其中，设置cv中的nfold为5折。最后输出这个方法找出的最优参数，得到n\_estimaters最优为2230。可以看到，当学习器的数目设置为2230时，使用80万条标记数据，10%作为训练数据，降维维度=300，实验结果如图4-7所示：

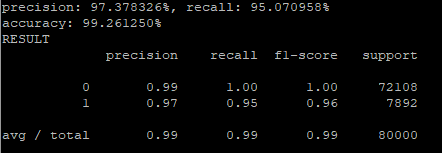


图4-7 n\_estimaters 2230

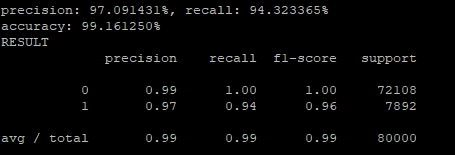


图4-8 n\_estimaters 1000

与n\_estimaters=1000对比，有明显的提升，尤其是在召回率上。、

## 4.4 优缺点分析

优点：训练效果好，不易过拟合。尤其是LGBM是GBDT的一个高效的实现，能够实现并行训练，有更快的速度和更低的内存消耗，且准确率更高。

缺点：相比于其他算法， 仍然速度较慢，内存占用仍较大。

# 5 SVM算法实现

## 5.1算法介绍

支持向量机（support vector machines，SVM)是一种二类分类模型。它的基本模型是定义在特征空间上的最大间隔线性分类器，间隔最大的条件是它与感知机的不同之处。支持向量机还包括核技巧，这使它成为实质上的非线性分类器。支持向量机的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。支持向量机的学习算法是求解凸二次规划的最优化算法。

支持向量机学习方法包含构建由简至繁的模型：线性可分支持向量机、线性支持向量机及非线性支持向量机。当训练数据线性可分时，通过硬间隔最大化，学习一个线性的分类器，即线性可分支持向量机，又称为硬间隔支持向量机；当训练数据近似线性可分时，通过软间隔最大化，也学习一个线性的分类器，即线性支持向量机，又称为软间隔支持向量机；当训练数据线性不可分时，通过使用核技巧及软间隔最大化，学习非线性支持向量机。

考虑一个二类分类问题。假设输入空间与特征空间为两个不同的空间。输入空间为欧氏空间或离散集合，特征空间为欧氏空间或希尔伯特空间。线性可分支持向量机、线性支持向量机假设这两个空间的元素一一对应，并将输入空间中的输入映射为特征空间中的特征向量。非线性支持向量机利用一个从输入空间到特征空间的非线性映射将输入映射为特征向量。所以，输入都由输入空间转换到特征空间，支持向量机的学习是在特征空间进行的。在实际的应用场景中，很多分类问题往往是非线性可分的，这时就需要用到核技巧的方法，常用的核函数有：线性核函数、多项式核函数、sigmoid核函数和高斯核函数。接下来我们也会比较这些核函数在该分类问题上的实验效果。

## 5.2实验过程

### 5.2.1 数据读取和存储

本实验所提供的短信数据共有100万条，其中带标签的数据80万作为训练和测试，不带标签的数据20万用于线上模拟。第一步需要从带标签的短信文档中，分割出文本域和标签域。作为content和label分别存入对应的json文件中。对于不带标签的短信数据，我们只需将文本域提取出来作为content存储到相应的json文件中，用于最终的线上模拟。

### 5.2.2 预处理与特征提取

本次实验使用jieba对数据进行分词，使用之前需要对一些非规范数据进行处理，如文档中多次出现替换隐私数字的xxxxxx，特殊字符&“】等进行统一转换。

对一些反复出现的无意义词汇，如‘的’等停用词进行筛选。

本实验采用了TF-IDF方式来构建词向量，这其中需要对TfidfVectorizer的参数根据实际情况进行调整。同时由于词向量数量庞大，为了便于后续处理我们先将其存储为csr型的稀疏矩阵到mtx文件中，便于后续模型的训练和测试。

### 5.2.3 降维

从分割后的训练集和测试集中取出的数据，通常都是高度稀疏的向量形式，即使是以csr稀疏矩阵的形式进行存储，在实际运算过程中也是转为高维向量的形式在进行计算。因此无论从节省时间，空间，还是减少数据噪声，提高分类精度的角度，降维过程都必不可少。在数据稀疏时，通常采用的降维方式包括了PCA(主成分分析法)，Random projections（稀疏随机投影）和Feature agglomeration（特征聚类）。这里以PCA为例，将训练集和测试集的维度降低到1000维。

### 5.2.4 SVM分类

scikit-learn中有着完备的关于SVM的算法，这里我们将其封装成一个训练器类，便于训练，调参，交叉验证以及在测试集上的评估。SVM根据其核函数的不同，可以分为各种不同类型，我们选取了四个核函数进行实验，分别为：线性核函数（linear），RBF核函数（rbf），多项式核函数（poly），sigmoid核函数（sigmoid）。

考虑到SVM的实际分类效果和硬件资源的限制，本实验只使用了10万条带标签的数据，其中9万条用于训练，1万条用于测试。四种核函数在测试集的结果如表5-1到表5-4所示：

表5-1 线性核

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| linear | precision | recall | F1 | support |
| 0 | 0.99 | 1.00 | 1.00 | 8974 |
| 1 | 0.97 | 0.95 | 0.96 | 1026 |
| Avg/total | 0.99 | 0.99 | 0.99 | 10000 |

表5-2 高斯核

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rbf | precision | recall | F1 | support |
| 0 | 0.99 | 1.00 | 1.00 | 8974 |
| 1 | 0.98 | 0.94 | 0.96 | 1026 |
| Avg/total | 0.99 | 0.99 | 0.99 | 10000 |

表5-3 多项式核

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| poly | precision | recall | F1 | support |
| 0 | 0.91 | 1.00 | 0.95 | 8974 |
| 1 | 0.97 | 0.15 | 0.26 | 1026 |
| Avg/total | 0.92 | 0.91 | 0.88 | 10000 |

表5-4 Sigmoid 核

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| sigmoid | precision | recall | F1 | support |
| 0 | 0.99 | 1.00 | 1.00 | 8974 |
| 1 | 0.98 | 0.94 | 0.96 | 1026 |
| Avg/total | 0.99 | 0.99 | 0.99 | 10000 |

因为垃圾短信类型分布的不平衡性，总体的评分本身没有太大的意义，因为将短信全部分为正常短信也能获得0.9以上的平均precision，recall和f1-score。这里我们主要关注的是1类型（垃圾短信类型）的precision，只有将垃圾短信正确的分类出来，模型本身才有意义，而不是将本身正常的短信划分为正常，可以看出1类型的precision表现还不错。整体的表现还是令人满意的，四个核函数中，除去poly外，非线性核函数和线性核函数的表现差异不大，但poly的性能劣于其他核函数，而且其训练时间也明显长于其他三个核函数。由于硬件资源的限制我们无法使用GridSearchCV方式来自动搜索最优参数，后续有条件的话会深入探讨。

### 5.2.5 交叉验证

由于SVM的时间和空间效率较低，本次实验只使用了10万条数据。训练数据较少的情况下，可以用 N折交差验证来一定程度的反映模型的准确性，同时可以做一个平均的度量，使得评价更佳稳定。这里交差验证的评价方式选择的是f1-score。

表5-5 交叉验证结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Cross-validation | 1 | 2 | 3 | 4 | 5 | Accuracy |
| linear | 0.9756 | 0.9760 | 0.9829 | 0.9751 | 0.9780 | 0.9775 |
| poly | 0.9372 | 0.9296 | 0.9444 | 0.9471 | 0.9421 | 0.9401 |
| rbf | 0.9722 | 0.9727 | 0.9789 | 0.9708 | 0.9759 | 0.9741 |
| sigmoid | 0.9727 | 0.9728 | 0.9789 | 0.9700 | 0.9745 | 0.9738 |

从预测结果来看，四个分类器都表现出较好的分类效果。其中poly由于其时间效率太低，没有得到其最优参数。从分类器各自角度来看，其在五次交叉验证过程中都表现出较好的鲁棒性。

从时间效率来看：线性核函数和sigmoid核函数较好，高斯核函数次之，多项式核函数最差，不适用于大规模数据的训练。

## 5.3优缺点分析

优点：SVM有较强的分类能力，尤其是其核函数分为线性和非线性两种，对于非线性问题可以通过非线性映射在线性空间解决问题。而且具有很好的鲁棒性。

缺点：该算法的时间和空间复杂度较高，对大规模训练样本难以实施。

# 6 决策树算法实现

## 6.1算法思想

决策树是附加概率结果的一个树状的决策图，是直观的运用统计概率分析的图法。机器学习中决策树是一个预测模型，它表示对象属性和对象值之间的一种映射。树中的每一个节点表示对象属性的判断条件，其分支表示符合节点条件的对象，树的叶子节点表示对象所属的预测结果。决策树模型常常用来解决分类和回归问题。常见的算法包括 CART (Classification And Regression Tree)、ID3、C4.5、随机森林 (Random Forest) 等。

## 6.2算法实现

下面我们使用决策树算法实现垃圾短信分类，分成以下几步进行处理。

### 6.2.1导入短信数据并分词

首先，我们需要对原始的有标签短信数据进行预处理，将内容和标签分别存为x和y，使用﻿json.dump将x和y存到本地。

文本的分类，基本上是基于词袋模型，也就是一个文本中包含多少词以及各个词的频率。对于英文，其天生的句子空格可以很容易的分割单词出来，但是中文就得先进行分词处理，也就是将一个完整的中文分割为一个一个词。

在Python中，有第三方模块——jieba，结巴分词来提供对中文的分词。我们使用jieba对短信的内容进行分词。

### 6.2.2分割训练集和测试集并提取文本特征

使用sklearn的分割模块对训练集和测试集进行分割。

从短信文本中提取特征，需要用到scikit-learn中的CountVectorizer()方法和TfidfTransformer()方法。

CountVectorizer()用于将文本从标量转换为向量，TfidfTransformer()则将向量文本转换为tf-idf矩阵。将生成的﻿data\_tfidf矩阵保存下来，用于模型训练。将﻿vectorizer和﻿tfidftransformer保存下来，用于模型预测。

### 6.2.3建立决策树分类器并进行训练

scikit-learn决策树算法库内部实现是使用了调优过的CART树算法，既可以做分类，又可以做回归。分类决策树的类对应的是DecisionTreeClassifier，而回归决策树的类对应的是DecisionTreeRegressor。两者的参数定义几乎完全相同，但是意义不全相同。

这里使用的是DecisionTreeClassifier，先从﻿sklearn中导入tree模块，﻿dt = tree.DecisionTreeClassifier()，使用﻿dt.fit(train\_x, train\_y)进行训练。

我使用了交叉验证 (cross-validation)的方法， 把数据分成数量相同的k=10 份，每次使用数据进行分类时，选择其中一份作为检验集，剩下的 k-1 份为训练集，重复 k 次，正好使得每一份数据都被用于一次检验集 k-1 次训练集。该方法的优点是尽可能多的数据作为训练集数据，每一次训练集数据和检验集数据都是相互独立的，并且完全覆盖了整个数据集。也存在一个缺点，就是分类模型运行了 k 次，计算开销较大。

## 6.3实验结果

本次实验，利用提供的80万带标签的短信，对其进行文本处理和特征提取，取其中80%为训练数据，其中20%为测试数据，用决策树模型进行训练。模型训练好之后，我们可以使用模型的predict()方法来测试与预测数据。因为之前对文本提取特征只是针对于训练集，测试集并没有进行，所以我先对测试集进行文本特征提取，再使用predict()方法进行预测。

scikit-learn模块中内置了很多模型测试、评估的方法，对于分类问题，我们可以使用下面几个方法：

模型训练时间用时：2280.024s

precision: 95.99%, recall: 95.08%

F1: 95.53%

accuracy: 99.11%

## 6.4优缺点分析及可能改进的方案

* 优点：1、需要很少的数据准备，其他技术通常需要数据标准化，需要创建虚拟变量，并删除空白值。2、能够处理数值和分类数据，其他技术通常是专门分析只有一种变量的数据集。3、可以使用统计测试验证模型。这样就可以解释模型的可靠性。4、即使它的假设在某种程度上违反了生成数据的真实模型，也会表现得很好。
* 缺点：1、决策树学习可能会生成过于复杂的树，不能代表普遍的规则。使用修剪机制，设置叶片节点所需的最小样本数目或设置树的最大深度。2、决策树可能不稳定，因为数据中的小变化可能导致生成完全不同的树。这个问题通过在一个集合中使用多个决策树来减轻。3、实际的决策树学习算法是基于启发式算法的，例如在每个节点上进行局部最优决策的贪婪算法。这种算法不能保证返回全局最优决策树。通过在集合学习者中训练多个树，可以减少这种情况，在这里，特征和样本是随机抽取的。
* 改进：过拟合是决策树的一大难点。我们知道决策树有多种算法，ID3，C4.5, C5.0, CART，sklearn库是使用的改良后的CART算法。我们可以通过多参数（max\_depth， min\_samples\_split，min\_samples\_leaf， min\_weight\_fraction\_leaf， max\_leaf\_nodes，min\_impurity\_split）设置，使用多个参数控制分叉的深度，防止过度拟合。

# 7 分布式机器学习算法实现

## 7.1环境准备

在笔记本上安装了两个Ubuntu 16.04版本的虚拟机，并实现了完全分布式的环境。篇幅有限，本章对搭建环境部分进行简要叙述，并叙述如何用Spark以及Scala实现分布式的逻辑回归算法。

首先安装两个虚拟机，系统版本Ubuntu 16.04。每个节点分配4G的内存（笔记本配置有限，故只实现两个节点的完全分布式，再多节点也一样）。并安装Java 1.8和Scala 2.12环境。安装过程和配置环境变量在这里不再叙述。

其中一个作为主节点，主机名为Hadoop1（下文以此称呼），另一个节点作为从节点，主机名改为Hadoop2。系统创建用户均为Hadoop。

生成主节点1的公钥发给从节点，生成从节点的公钥发给主节点以实现互相ssh免密码登录。并固定Hadoop1的ip为192.168.8.6,Hadoop2的ip为192.168.8.7。

## 7.2 Hadoop环境搭建

在这一节，简要概括Hadoop完全分布式环境的搭建过程，不再大篇幅给出配置文件的细节，只叙述流程。过程如下：

1. 下载Hadoop2.7.3二进制包；
2. 将二进制包解压到/opt目录下，并在/etc/profile里配置相应环境变量；
3. 将Hadoop-env.sh脚本中修改默认的Java的环境路径，并在slaves文件里输入对应工人节点的主机名；
4. 分别对core、hdfs、yarn等配置文件进行修改；
5. 修改好配置文件后，格式化一下namenode；
6. 将/opt下的修改好配置的Hadoop文件夹远程拷贝到从节点下，同样修改/etc/profile如主节点。

在按上述流程配置好之后，可以在主节点通过Hadoop/sbin/start-all.sh启动节点进程。启动后，可以通过jps查看进程是否启动成功。如图7-1所示：

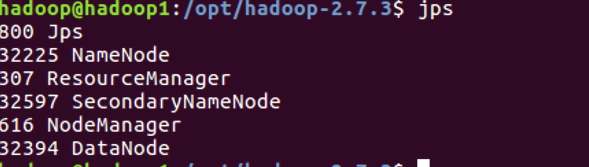


图7-1 主节点进程

在主节点启动进程成功后，可以在从节点Hadoop2查看，如图7-2所示：

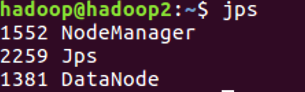


图7-2从节点进程

一般这样就是搭建好了，可以通过Hadoop fs -mkdir /Hadoop在hdfs里创建/Hadoop文件夹，并提交任务进行测试。我搭建后提交了wordcount任务进行测试，发现Hadoop完全分布式的环境搭建成功。

## 7.3 Spark环境搭建

在搭建好Hadoop环境后，已经可以实现分布式。这次大作业主要通过Spark MLlib来实现分布式的机器学习分类算法，所以还需要搭建Spark环境。Scala环境已经提前搭建好，接下来搭建Spark环境。依旧流程如下：

1. 下载spark-2.1.0-bin-hadoop2.7包；
2. 解压后依旧移动到/opt文件夹下；
3. 修改系统配置文件添加spark的环境变量；
4. 修改Spark的spark-env.sh,对应添加Scala、Java、Hadoop的环境变量，并指定主节点的ip和worker\_memory(这里就和Hadoop连了起来)。并在slaves里指定工人节点主机名;
5. 将配置好的spark文件夹拷贝到从节点中，进行相应的系统环境变量配置。

到这，Spark环境基本配置好了。在启动Hadoop进程的基础上，在Spark文件夹下的通过sbin/start-all.sh 启动spark进程。启动后，调用jps查看建成，如图7-3所示:

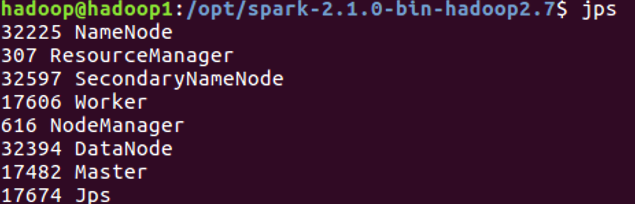


图7-3 主节点Spark进程

从图7-3可看出，启动spark脚本后，主节点多了master和worker进程。在从节点Hadoop2调用jps，如图7-4所示：

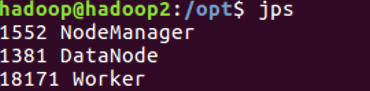


图7-4 从节点Spark进程

搭建好Spark后，可以使用Spark-shell命令进入spark-shell，对简单的程序进行测试。

进入spark-shell后，输入如下简单的wordcount程序，测试是否搭建成功。

## 7.4 Spark MLlib逻辑回归实现

在Ubuntu下下载Intellij IDEA进行编程，建立Scala工程。Spark下主要通过Mlib的classification.LogisticRegressionWithSGD包来实现逻辑回归算法。逻辑回归原理在前面已经有了简要叙述，这里不再叙述，主要介绍如何实现分布式的。

Spark主要通过RDD（弹性分布式数据集）来实现逻辑上的并行数据处理。数据读入后构建RDD。

RDD提供了一种高度受限的共享内存模型，即RDD是只读的记录分区的集合，只能通过在其他RDD执行确定的转换操作（如map、join和group by）而创建，然而这些限制使得实现容错的开销很低。对开发者而言，RDD可以看作是Spark的一个对象，它本身运行于内存中，如读文件是一个RDD，对文件计算是一个RDD，结果集也是一个RDD ，不同的分片、 数据之间的依赖 、key-value类型的map数据都可以看做RDD。

在实际实现时，将数据放入RDD中，Spark就会根据节点情况（这也取决于不同模式）来进行调度，对RDD的数据进行分布式处理。值得注意的是，在机器学习算法中，训练数据要进行迭代，所以在迭代之前一定要通过cache方法将数据加载到内存中，这样方便迭代运算。

以下举部分构造训练集的程序来说明实现过程。

将读入的垃圾短信的文本进行分词后，进行去停用词等处理后放入rdd2中。

val spamFeatures = rdd2.filter(\_.\_2.equals("0")).map(text => tf.transform(text.\_1))

val hamFeatures = rdd2.filter( \_.\_2.equals("1")).map(text => tf.transform(text.\_1))

通过如上的程序来将rdd2的数据分别映射到垃圾短信的特征rdd中和非垃圾短信特征中。

val positiveExamples = spamFeatures.map(features => LabeledPoint(1, features))

val negativeExamples = hamFeatures.map(features => LabeledPoint(0, features))

val trainingData = positiveExamples.union(negativeExamples)

trainingData.cache()

以上将特征对应好标签，并形成训练集。并将训练集加载到内存中。

最后训练输出结果，并将结果存入到hdfs://192.168.8.106:9000/Hadoop/Output/result里。

## 7.5 Spark MLlib实验结果

实现逻辑回归的程序后，将任务提交进行实验，为了方便分布式的对数据进行读取，所以将输入文件上传到hdfs中。

先将编写好的程序打包成test2.jar，依赖包可以部署到Spark的lib里也可以在提交任务时，指定依赖包。

通过spark-submit命令提交任务到分布式系统中，如图7-5所示：



图7-5 提交spark任务

提交任务后，程序开始运行，登录localhost：8080可以查看节点情况以及查看任务执行情况。如图7-6所示，页面上方显示了主节点的ip和端口，以及节点数量。中间是节点情况表格，显示节点的名称，ip，以及所有内存。

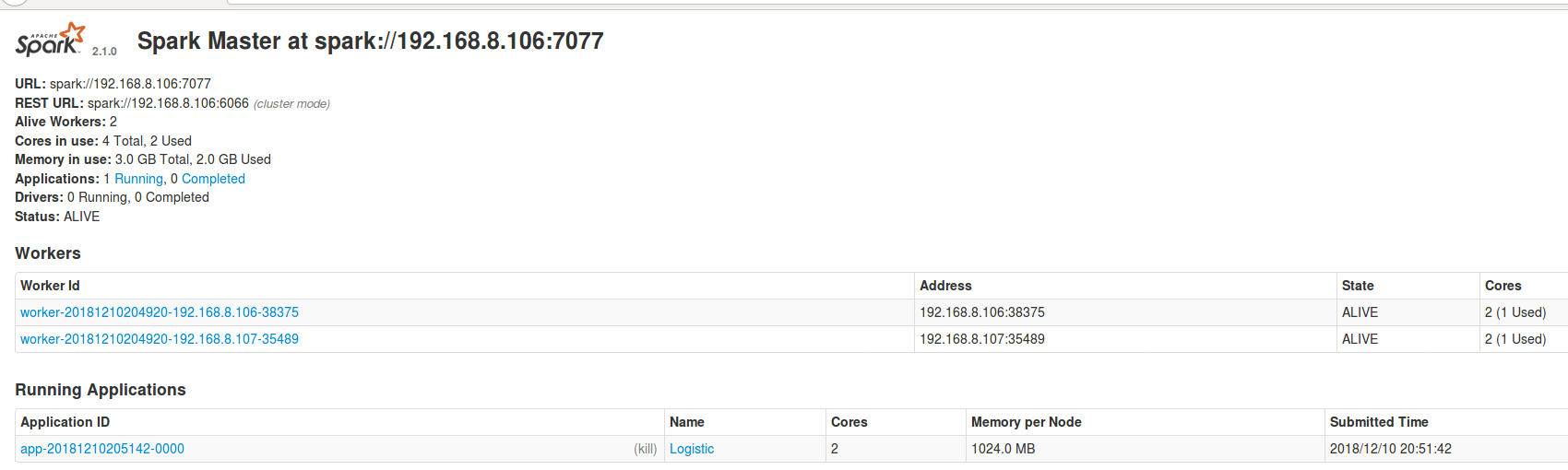


图7-6 Spark UI页面

可以看到此时正有着一个logistic任务在运行，在任务运行结束后，会出现在completed Application里。点击任务的id可以进到任务详情页面，如图7-7所示。

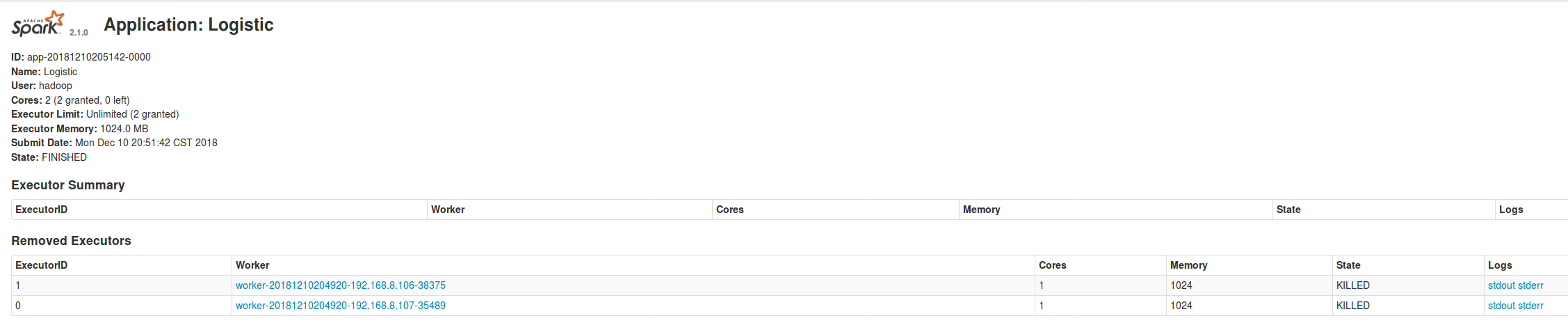


图7-7 Application 页面

在该页面可以看到任务的提交日期，以及任务状态，节点状态。同时可以查看任务的输出日志。同样的，在localhost：4040可以查看正在运行的任务的监控情况，但任务执行时间较短（由于内存限制没有跑全部的数据），所以没有什么有意义的监控指标。

在任务完成后，通过hadoop fs -ls /Hadoop/Output/result命令可以看到，成功将结果输出到hdfs中的result文件夹里。会发现有三个文件，分别是\_SUCCESS,part-00000,part-00001。由于有两个节点参与运算，所以在运算过程中，会将输入测试集分为两批进行预测，结果分别在00000和00001中各占一半。如图7-8和图7-9所示。

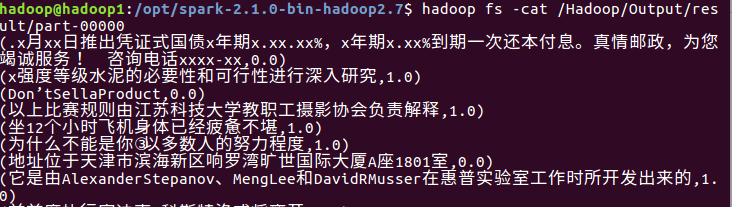


图7-8 part-00000文件部分

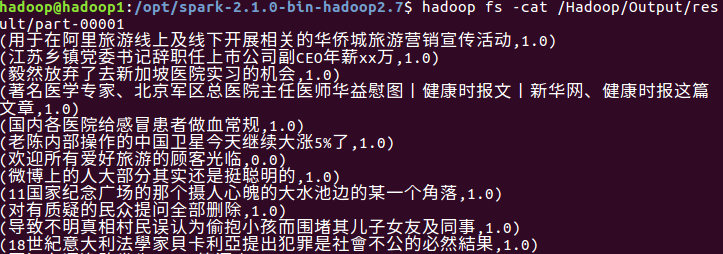


图7-9 part-00001文件部分

## 7.5 结论分析

在亲手实现分布式的机器学习算法后，对算法如何分布化执行的有了更深刻的认识。Spark MLLib通过rdd的实现，能够对数据进行划分和映射。

算法的分布化的确提高了算法的计算速度，但这也是一个拿空间换时间的过程。因为在不同节点之间，需要进行同步和通信，所以会占用大量内存。具体表现在Spark中就是同样的数据，在处理过程中map的RDD越多，就会导致内存占用越大。所以在编写Spark程序时，需要注意，能够减少Rdd的生成，就尽量避免新的rdd的生成，能够复用rdd就复用rdd。

另外，机器本身的进程启动和通信也占用一部分时间，所以如果数据很少，通信占用的时间就能凸显出来。最后对于机器学习来说，在迭代训练之前，要把训练数据加载到内存中，方便迭代时调用，以提高效率。

# 8 可视化系统设计与实现

## 8.1开发环境

操作系统： Ubuntu

内存： 2G

Python版本：Python 3.5.0

Django版本：Django 2.1.3

Mysql版本： Mysql 5.7.24

## 8.2系统结构

系统结构如图8-1所示：



图8-1可视化系统结构图

## 8.3整体设计

整体系统主要通过基于Python开发的Django框架搭建。Web界面设计中使用HighCharts组件，HTML和JavaScript脚本等技术。数据库设计中涉及Mysql数据库，Navicat连接管理。算法模块引用了Tensorflow、sklearn、Jieba等包。

### 8.3.1 垃圾短信分类模块

从前端界面提交待测试文本，以及选择算法的标号。提交至views.py，由该文件中的响应函数调用具体的算法模块，返回测试结果和测试时间。

### 8.3.2 算法指标展示模块

Web界面中的展示图表是基于HighCharts组件开发， JavaScript脚本定义图表样式，以及接收四个算法的不同指标值。

## 8.4数据库模块

### 8.4.1 前端展示

数据库展示界面由Django中的站点管理接口admin实现。首先，定义界面数据表中需要展示的字段。同时注明可以编辑的字段，以及不同颜色展示的字段，让垃圾信息和正常信息区别明显。如下图8-2所示。

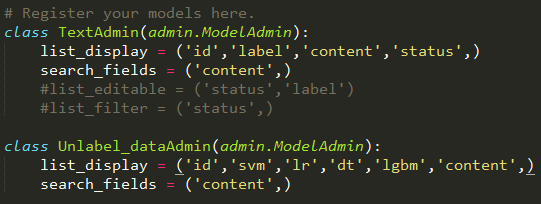


图8-2 不同信息区别

在models中定义数据库表的详细结构，以下图无标签数据为例。

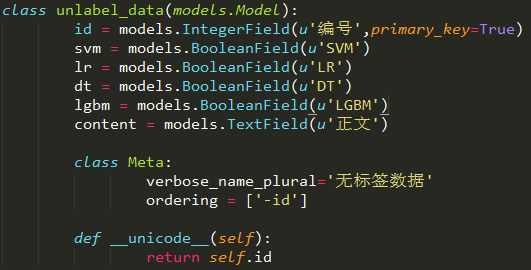


图8-3 无标签数据

### 8.4.2 后端存储

首先，将不同算法对无标签数据的测试结果进行整合，连同原来的有标签数据一同存入数据库。可以通过程序导入，但是考虑实现效率问题，可以通过Navicat工具将两个txt数据文件导入Mysql数据库，分别是text和unlabel\_data表。

如图8-4所示，experiment\_text表有标签数据进行了存储和整合，方便可视化系统进行读取。

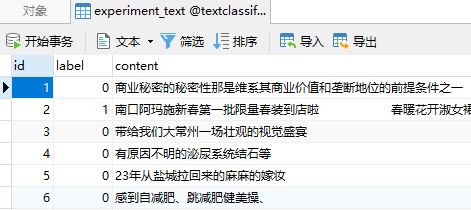


图8-4 experiment\_text表

Unlabel\_data表对未标签数据进行存储和读取，如图8-5所示。

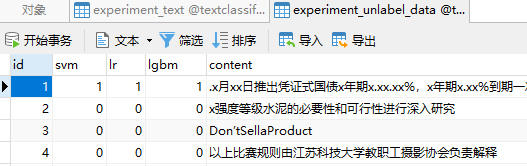


图8-5 unlabel\_data表

## 8.5算法模块

首先，由算法选择函数接收由前端发送的算法标号，并调用具体的算法模块执行分类程序，并返回各个算法的分类结果和分类时间。

如图8-6是对所有算法的整合调用。

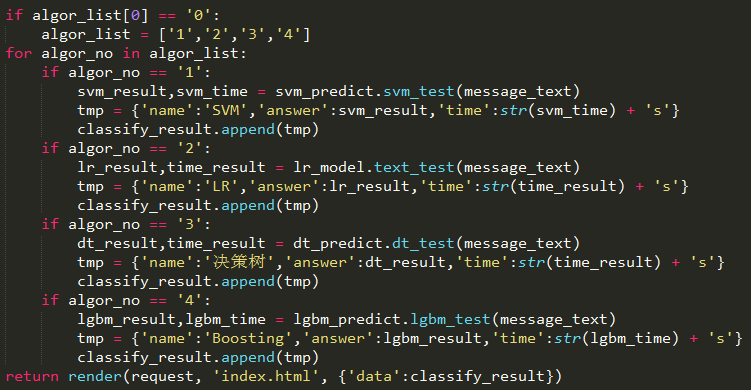


图8-6 算法整合和调用