集成学习 实验报告

计82 郑凯文 2018011314

实验简述

实验的基分类器有决策树、SVM两种，数据集为Amazon商品评论，目标为从文本预测rating（1~5共5个类别）。在本次实验中，我实现了Boosting、Adaboost两种集成学习方法来解决多分类问题，并进行了数据处理、特征选取、参数调节等充分的实验。

代码说明

- `split_dataset.py` 直接运行，解析源数据并进行预处理、数据集的划分
- `get_features.py` 直接运行，从划分的数据集提取特征。在代码中可修改使用tf-idf特征，或count特征。
- `train_test.py` 直接运行，训练模型并给出测试集上的准确率、MAE、RMSE指标。代码中可修改使用的基分类器、集成方法等。

数据处理与特征提取

数据集处理

数据集截取

数据共220000条，当特征维数较大时，决策树的训练十分耗费时间，且集成方法需要成倍的训练数量，这使得总体训练过慢。为了加速训练，我截取了1/10的数据，即22000条进行训练和测试。

预处理与去噪

源数据是具有噪声的，如混杂着标点符号、颜文字（如^_^）、数字等。我进行实验的目的是根据英文文本对rating进行预测，因此需要去除无关内容。我进行的预处理有

- 去除标点符号、占位符
- 使用空格分隔成词列表，并去除空文本

我还进行了去除停用词与否的测试。停用词一般是信息量低、出现频率高的词，去除后有利于节省空间和提升训练效果。我使用的是`nltk`库中的停用词表。

特征选择

在数据集中，与rating相关的文本有`summary`、`reviewText`两项。我使用的方法是：优先采用`summary`作为待解析的文本，若其中无法解析出任何有用信息，再使用`reviewText`进行解析。这样做的原因是，`summary`往往短小精炼、感情色彩浓厚，有很显著的词如nice、excellent，不但有利于减小词表，还有利于分类。

数据集划分

我采用俄罗斯轮盘赌方法，将22000条数据按照训练集:测试集=9:1进行划分。预处理、划分后的数据集使用python的json库进行序列化并保存在文件中，便于特征提取时加载。

文本特征提取

由于本次实验不提倡使用词嵌入，我使用传统方法来提取特征向量，一个是简单地将频数作为特征，另一个是tf-idf特征。

频数截取

频数和tf-idf特征的向量长度都和词表大小有关，而训练中不同的词有数万到数十万，使得特征矩阵过大。我只保留频数最大的一些词作为词表，并对不同的截断长度进行实验。

在实现中，首先用`sklearn`的`counter = CountVectorizer(max_features=xxx)`得到全局词表，之后进行特征提取时均设定`vocabulary=counter.vocabulary_`参数。

Count

频数特征向量的每个位置是对应词在文本中的频数。

在实现中，可以直接使用`sklearn`的`CountVectorizer().transform()`得到。

TF-IDF

tf-idf是词频（tf）和逆向文件频率（idf）的乘积。包含某个词的文档越少，则idf越大，因此tf-idf相较于词频，能给一些比较"稀有"的词更高的权重。

在实现中，使用`sklearn`的`CountVectorizer,TfidfTransformer`进行提取。

特征保存与加载

对提取得到的特征调用`toarray()`可以得到numpy矩阵，之后保存为npy文件。npy文件可以方便地被numpy加载，在训练时可直接加载特征矩阵。

模型设计

基分类器

决策树

决策树是一种非参数方法，具有较好的可解释性，可以构造较为复杂的决策边界。实验中，在特征向量长度上千时，决策树相较于线性SVM具有明显较慢的训练速度。

实现时，使用`sklearn.tree`库。

SVM

由于特征向量较长，在使用非线性核函数（如rbf）时，训练极慢，几乎无法进行。我采用的是`sklearn.svm`中的`LinearSVC`，因此只要特征具有很好的线性可分性时，表现才比较出色。

集成方法

Bagging

Bagging每次对训练集进行有放回的采样（bootstrap），得到新训练集进行训练。对多次bootstrap训练得到的分类器，进行多数投票的方式决定最终的分类。

在实现中，可使用`np.random.choice(length, length, replace=True)`进行有放回采样。

AdaBoost

我实现的是AdaBoost M1方法，它和AdaBoost的不同只在于是否当基分类器错误率较高时退出。而在我的实现中，AdaBoost进行若干步，便会达到错误率阈值而退出。

为解决这个办法，我查阅了文献Multi-class AdaBoost。对于有K类的多分类，其中SAMME算法的解决方案是：

- 将错误率阈值更改为$(K-1)/K$，也就是随机分类的错误率，这样就不会中途退出了
- 更改权重修正项及投票权重。对应于AdaBoost M1方法，就是将原先β的计算公式除以$K-1$

我对这些修正进行了实验，并分析效果和原因。

实验与分析

我首先对数据处理方法进行实验和调节，只选用SVM作为基分类器，而不使用集成方法。

使用停用词

在频数截断为100、使用tf-idf特征时，停用词的影响如下：

使用停用词	准确率
否	61.0%
是	61.3%

可见，使用停用词会略微提高准确率，这是由于有限的特征向量长度下，停用词可以去除一些冗余信息，如介词for、动词is等。

特征选取

在频数截断为100、不使用停用词时，我测试了频数、tf-idf两种特征的效果

特征	准确率
count	60.2%
if-idf	61.0%

准确率存在一定波动，但总体tf-idf更优。且在使用频数特征时，SVM训练中经常出现警告：`ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.`。这说明tf-idf相较于频数，更有利于线性分类，更快收敛。这是由于tf-idf相较于词频，能给一些比较"稀有"的词更高的权重，而这些"稀有"的词往往具有较好的类别区分能力，可以很好地代表类别特征，使得分类更容易了。

频数截断

在使用停用词、tf-idf特征时，我测试了不同的频数截断长度对准确率的影响。

长度	准确率
100	61.3%
500	62.7%
1000	62.7%

增大特征长度可以提高准确率，而达到一定长度后提升便不明显了。这可以从以下数据解释：当词表大小为100时，有28%的样本中的词均未在词表中出现，即特征向量全0。而词表大小为500、1000时，这个数字是11%、7%，可以忽略。增大特征向量长度，也就是增大词表，可以保留更多文本信息，进行更加特化的分类。

当然，这也是由于tf-idf向量长度受限于词表的问题。因此，采用词嵌入等作为特征向量或许是更好的选择。

Bagging：采样覆盖率

关于Bagging，一个思考的问题是，进行多少次bootstrap，所有采样出的数据集可以覆盖整个数据集？这个问题可以从理论上和实践上分析。

考虑1次bootstrap，采样的数据集能覆盖整个数据集的多大部分。设数据集大小为N，则某个样本被选中至少1次的概率是$1-(1-1/N)^N$。当N比较大时，大约是$1-1/e=0.63$。在实验中，我进行统计并输出，每次覆盖的比例大约是$12400/19768 \sim 12500/19768$，也就是$0.627 \sim 0.632$。理论和实践符合很好。

这样，假设进行M次的Bagging，被覆盖的比例大约是$1-1/e^M$。由于是指数减小，未被覆盖的比例下降很快。在实验中我选取$M=10$，此时超过99.99%的数据集被覆盖。

AdaBoost：样本权重vs重采样

在AdaBoost中，需要给样本赋权，而一些问题不容易处理权重。一个简便的方法是，按照权重作为概率，对训练集进行有放回的重采样，这样能达到和赋权类似的效果。前者更准确，而后者更方便

我实现了这两种方式：

样本权重：

```
model.fit(f_train, y_train, sample_weight=weights * length)
```

重采样：

```
indices = np.random.choice(length, length, replace=True, p=weights)
model.fit(f_train[indices], y_train[indices])
```

经测试，两种的效果是差不多的。

AdaBoost：多分类改进

在AdaBoost中，我总是遇到算法提前终止的问题。如之前所述，我采取了三种措施进行对比：

原版AdaBoost：（raw）

```
if et > 1 / 2:
    break
beta = et / (1.0 - et)
```

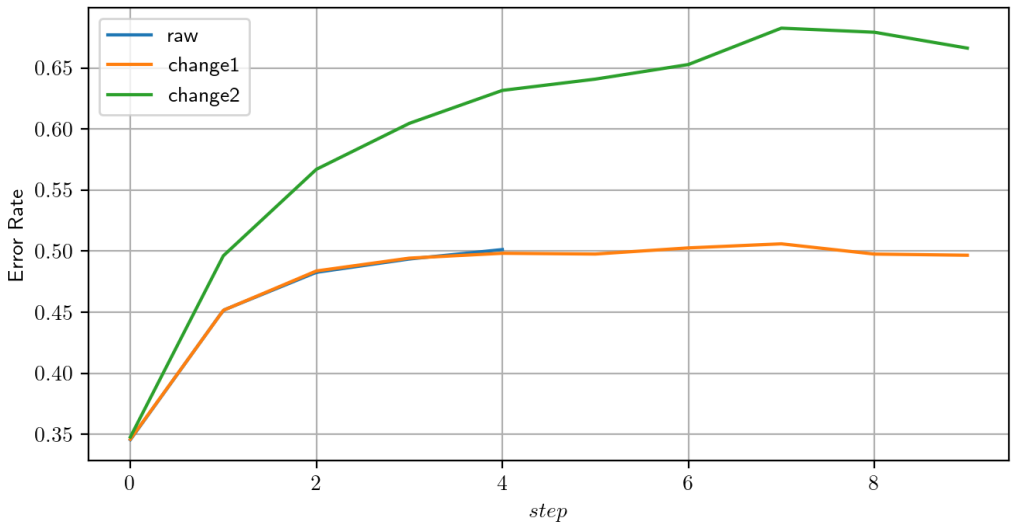
改变错误率阈值：（change1）

```
if et > 4 / 5:
    break
beta = et / (1.0 - et)
```

改变错误率阈值+改变权重调节：（change2）

```
if et > 4 / 5:
    break
beta = et / (1.0 - et) / 4
```

采用SVM、tf-idf、使用停用词、特征向量长度为1000，AdaBoost迭代10步，三者每步的错误率变化如下：



对于raw版本，错误率很快超过50%而终止。对于change1，容忍更高的错误率，而不改变权重调节方式，使得错误率维持在50%附近（因为错误率>50%时，便会增加正确分类的样本权重）（这相当于不带中退的AdaBoost而非AdaBoost M1）。对于change2，二者都改变，因此错误率朝着80%增加。

最终的准确率如下：

策略	准确率
raw	62.1%
change1	62.1%
change2	56.7%

从效果来看，raw和change1类似，这是因为之后的几个基分类器，错误率都在50%附近，它们的投票权重接近于0，有没有影响不大。而出乎意料的是，change2会使准确率大幅下降。对此，我的猜测有：

- 类别不均衡，如全预测5也有60%左右准确率，这使得每个类别先验概率不同，应用于平衡多分类的修正不再适用
- 特征信息不充分、基分类器太弱（上面也提到过，这里只使用了线性SVM，对特征精炼度有很高要求），因此后面训练出的、错误率超过50%的基分类器价值不大，甚至起反作用。

最终结果与总结分析

综上，我使用tf-idf作为特征、使用停用词、取特征向量长度为1000。对于AdaBoost，我还是选用了raw版本，只能说这个应用场景里，它更适合一些。对于Bagging而言，$M = 10$。最终的结果如下：

组合	准确率	MAE	RMSE
TREE	56.7%	0.7004	1.2379
SVM	62.0%	0.6296	1.1851
TREE+Bagging	60.3%	0.6144	1.1372
TREE+AdaBoost	59.7%	0.6288	1.1525
SVM+Bagging	63.7%	0.6068	1.1704
SVM+AdaBoost	62.2%	0.6261	1.1836

从中有以下结论：

- Bagging和AdaBoost均能提升基分类器的性能。对于决策树，这种提升尤为明显，这时由于决策树具有如下特点：单颗决策树灵活、对噪声鲁棒，在训练集上拟合很好（在AdaBoost过程中，决策树没有中途退出，而SVM有，这说明决策树的拟合能力强于线性SVM），但很难泛化（也就是容易过拟合）。因此，决策树十分适用于集成方法，一种很常见的做法就是构建很多小的、弱的决策树分类器，将它们集成起来。对于比较稳定的分类器如SVM，集成学习方法提升较小。
- 在我的实现中，AdaBoost性能不如Bagging，这可能是因为AdaBoost不太能容忍噪声，且需要比较强的基分类器。而Bagging较为稳定，大多数情况下都可以稳定地提高性能。

综上，使用SVM+Bagging的组合、tf-idf特征、去除停用词、频数截断长度为1000的效果是比较理想的。另一方面，这个数据集上，全部预测5也有将近60%准确率，这说明实验中的方法改进并不明显。进一步地，需要采用更强的特征提取方法，且自然语言处理的一些内容会派上用场，如一些summary中直接写"5 stars"，这些都是可以利用的信息。