实验3-集成学习实验报告

陈意扬 计96 2019011341

实验目的

1）实现`Bagging`和`AdaBoost`算法并在给定数据集上测试

2）比较不同基分类器和不同集成学习算法的组合

3）分析实验结果

实验原理

`Bagging`算法步骤：

$For\ \ t=1，2，\ldots，T\ \ Do$
　从S中拔靴采样产生D_t
　在D_t上训练一个分类器H_t
分类一个新的样本 $x\ \epsilon\ X$ 时，通过对H_t多数投票（等权重）

拔靴法/自举法采样指均匀随机的有放回采样。`Bagging`算法的关键在于对`T`的选择和采样的比例。

`AdaBoost`算法步骤：

初始给每个样本赋相等权重为$1/N$;
$For\ t=1,2,\ldots,T\ Do$
　1.生成一个假设C_t
　2.计算错误率 ϵ_t：ϵ_t = 所有错误分类的样本权重和
　3.计算投票权重α_t：$\alpha_t=\frac{1}{2}ln\frac{1-\epsilon_t}{\epsilon_t}$（二分类）
　4.更新每个样本的权重：
　　正确分类：$W_{new}=W_{old}*e^{-\alpha_t}$
　　错误分类：$W_{new}=W_{old}*e^{\alpha_t}$
　5.归一化权重（权重和=1）
融合所有假设C_t，各自投票权重为α_t

经查阅资料根据`sklearn`中的`SAMME`算法，多分类的投票权重为
$\alpha_t=learning\_rate*(ln\frac{1-\epsilon_t}{\epsilon_t}+ln(R-1))$

`AdaBoost`算法的关键在于对于`T`的选择。

实验步骤

1.文本向量化

我采用了2种文本向量化方法：
　1.较为简单的`BOW`；
　2.`word embedding`方法，具体是`gensim`库中的`Word2Vec`。

提供的数据集共有22w条对商品的打分以及简评，存储在`exp3-reviews.csv`中，每个评论分为`overall`、`reviewerID`、`asin`、`unixReviewTime`、`summary`和`reviewText`6栏，我提取`overall`部分的打分作为label，并将`summary`和`reviewText`这两栏作为特征进行向量化。

BOW（parse_csv_bow.py）

首先对文本进行分词和去停用词，这次去停用词时我保留了所有形容词和程度副词。我把`summary`和`reviewText`合并作为一个文本。之后统计词频，取词频前1k的词，每一个词代表1k维向量的一维。再对文本进行向量化，对于文本中的每一个词在向量对应维度+1，这样每一个文本就能对应一个1k维向量。值得注意的是我对数据集中label的分布进行了统计

```
, ('walk', 1876), ('stock', 1871), ('key', 1869), ('brought', 1863)]
1000
{1: 10134, 2: 10168, 3: 21960, 4: 48672, 5: 129066}
root@da272ac67498:~/thu-machine-learning-2022/ensemble-learning#
```

可以发现各个label的分布极不均匀，为了避免高词频的词全部出自5分条目，我对不同label提取的词进行了加权处理

```
dic[word]+=5.0*(rating==1)+5.0*(rating==2)+2.5*(rating==3)+1.0*(rating==4)+1.0*
(rating==5)
```

使特征词相对均匀。

Word2Vec（parse_csv_word_embedding.py）

同样对文本进行分词和去停用词后，我将`reviewText`拆分成句子（`summary`作为1个句子），将句子集进行`word2vec`训练，将每个词训练成50维的向量（去掉词频低于10的词）。`vocabulary`中包含5w左右个词。同样把`summary`和`reviewText`合并作为一个文本后进行向量化，首先进行长度统计

```
100   # print(model.wv[0])
101   # for i in range(len(model.wv)):
102   #     np_wordList.append(model.wv[i])
103   # np_wordList.append(complement_vec)
104   # print("words: "+str(len(np_wordList)))

问题  输出  调试控制台  终端                                zsh - e

219994
219995
219996
219997
219998
219999
220000
sentences: 1721921
max len: 2733
len200+: 9556
len150+: 16857
len100+: 32496
len50+: 75059
len30+: 116150
len20+: 152065
➜  ensemble-learning git:(main) ✗

行 81，列 21   空格: 4   UTF-8
```

长于100词的文本不到15%，所以我选择选择100词作为标准长度，对文本按词出现的顺序将每个词对应的50维向量首尾相接成一个5000维向量（不到100词的文本用0向量填充，超过100词的文本进行截断）。这样每一个文本就能对应一个5k维向量。`word2vec`相对于`BOW`的好处在于能够包含一定的语义信息，同时保留了文本中词出现的顺序信息，但缺点在于开销更大，同时很难用低维向量表示文本。

2.数据集切分(train_test.py)

我将所有的条目`shuffle`之后取前20w作为训练集，后2w作为测试集。

3.模型训练

`SVM`（`svm.py`、`svm_bow.py`）

使用`sklearn`库中的`svm.SVC`模型进行训练，`decision_function_shape`选择“`one vs rest`”（多分类），`kernel`选择“`rbf`”。`Word2Vec`表示的5k维向量最大迭代次数设置为100，`BOW`表示的1k维向量最大迭代次数设置为500。

`DecisionTree`（`dt.py`、`dt_bow.py`）

使用`sklearn`库中的`tree.DecisionTreeClassifier`模型进行训练。

`Bagging-SVM`&`Bagging-DecisionTree`（`bagging_svm.py`、`bagging_svm_bow.py`、`bagging_dt.py`、`bagging_dt_bow.py`）

拔靴采样我的实现是每次对训练集进行shuffle之后取前2w、4w和8w的条目进行训练（实际上是shuffle序号，这样开销比较小）。T的选择我选择了10、20和30。每个分类器的训练和上述相同。

`AdaBoost-SVM`&`AdaBoost-DecisionTree`

$\alpha_t = learning_rate * (ln\frac{1-\epsilon_t}{\epsilon_t} + ln(R-1))$中的学习率我选择了0.5，T的选择我选择了5、10和15。

模型的评价（/results）

（所有的实验结果数据都可以在`/results`目录下找到截图）

由于是多分类，我采用了2个评价指标：

1.准确率`acc`；

2.预测`label`和`ground truth label`的平均差异：`average difference`，简称`ad`。

基分类器

	`acc;ad`
$SVM_{Word2Vev}$	25%；2.15785
SVM_{BOW}	42%；1.45385
$DT_{Word2Vev}$	48%；0.86885
DT_{BOW}	51%；0.80545

就基分类器来说，`DecisionTree`分类器在这个分类任务上表现优于`SVM`分类器，而`BOW`文本向量化方法表现优于`Word2Vec`方法。对比两种文本向量化方法得出的实验结果`SVM`随向量维度提升性能下降明显，`DT`则受到有限的影响。

集成学习模型调参后表现

	base	Bagging	AdaBoost
$SVM_{Word2Vev}$	25%；2.15785	35%；1.74215	36%；0.947
SVM_{BOW}	42%；1.45385	48%；1.0614	43%；1.4632
$DT_{Word2Vev}$	48%；0.86885	60%；0.6797	55%；0.71715
DT_{BOW}	51%；0.80545	61%；0.64745	56%；0.7107

1.增益：分析数据可知集成学习方法对所有基分类器均有增益效果，其中`Bagging`方法在本分类任务上对基分类器的增益更加显著和稳定，准确率的增益在14-40%，平均增益23%；`AdaBoost`在基于`BOW`的`SVM`分类器上几乎没有增益，但在在基于`Word2Vec`的`SVM`分类器上又有非常出色的表现（44%增益），平均增益14%。

2.性能：基于`DT`的`Bagging`集成学习模型 > 基于`DT`的`AdaBoost`集成学习模型 > 基于`SVM`的`Bagging`集成学习模型 =?= 基于`SVM`的`AdaBoost`集成学习模型

DT-Bagging	DT-AdaBoost	DT-base	SVM-Bagging	SVM-AdaBoost	SVM-base
61%；0.65	56%；0.7	50%；0.8	42%；1.4	40%；1.2	34%；1.8

参数对模型的影响

Bagging

T（sample = 2w）

T	10	20	30
$SVM_{Word2Vev}$	35%；1.74215	34%；1.7455	35%；1.7464
SVM_{BOW}	48%；1.0614	48%；1.06945	48%；1.0637
$DT_{Word2Vev}$	58%；0.70105	59%；0.68775	60%；0.6797
DT_{BOW}	59%；0.66425	60%；0.6511	61%；0.64745

分析实验数据，参数T对于`SVM`的影响不大，对`DT`有较为有限的影响，随T增大`DT`获得稳定的极小幅增益。我认为当T达到一定值后训练集中所有样本都有极大概率取到一次（T=10->65%，20->88%，30->96%），集成学习模型的表现也会变得稳定。

sample（T=10）

sample	2w	4w	8w
$SVM_{Word2Vev}$	35%；1.74215	32%；1.8579	29%；2.0108
SVM_{BOW}	48%；1.0614	46%；1.2898	44%；1.37715
$DT_{Word2Vev}$	58%；0.70105	59%；0.68145	59%；0.67475
DT_{BOW}	59%；0.66425	60%；0.65615	59%；0.653

分析实验数据，sample量对`Bagging-DT`几乎无影响，而随sample量增加`Bagging-SVM`性能明显下降。我推测`DT`对于数据集规模较为不敏感，而`SVM`在低维小规模数据集上表现较好而随数据集规模增大分类效果会明显变差，于是集成的基分类器随sample量增大而变差，从而导致集成学习分类器变差。

`AdaBoost`

T

T	5	10	15
$SVM_{Word2Vev}$	35%；0.95425	36%；0.947	36%；0.947
SVM_{BOW}	43%；1.4632	43%；1.4632	43%；1.4632
$DT_{Word2Vev}$	53%；0.7724	54%；0.7367	55%；0.71715
DT_{BOW}	54%；0.7345	55%；0.7183	56%；0.7107

分析实验数据，参数T对于`SVM`的影响不大，对`DT`有较为有限的影响，随T增大`DT`获得稳定的极小幅增益。我进一步查看了SVM随着T增大的模型的投票权重的变化，如下图

```
[0.12247876, -0.32302598, 0.14103, 0.07053526, -0.69116896, -0.34558448, -0.17279224, -0
.04579762, -0.09163305, -0.04579939, -0.02291698, -0.01145849, -0.00572924, -0.00286462,
 -0.00143231]
acc: 36%
average difference: 0.947
root@da272ac67498:~/thu-machine-learning-2022/ensemble-learning#
[ada-svm] 0:bash*                                    "da272ac67498" 04:24 07-May-22
```

可见SVM对于加权样本的训练效果较差，错误率提升很快，让模型很快达到饱和，后面模型的权重接近于0，从而不会影响投票结果。相对的，DT对于加权样本的训练效果较好，错误率提升慢，让模型达到饱和所需迭代轮数较多，随模型的增加可以获得小幅增益。

实验结论

对于本次分类任务

性能上基于`DT`的`Bagging`集成学习模型 > 基于`DT`的`AdaBoost`集成学习模型 > `DT`基分类器 > 基于`SVM`的`Bagging`集成学习模型 =?= 基于`SVM`的`AdaBoost`集成学习模型 > `SVM`基分类器。

性能增益上`Bagging`集成学习算法 > `AdaBoost`集成学习算法。

参数上`Bagging`集成学习算法和`AdaBoost`集成学习算法的迭代轮数T对性能的影响较小，而采样规模对于基于`Bagging-SVM`的性能影响极大，我认为是SVM算法本身对数据规模的敏感性导致。