

# Machine Learning Exp2 Report

---

2016011395 石景宜 kaggle id:sjy2016011395

## 一、实验背景

---

- 集成学习：使用一系列学习器进行学习，并使用某种规则把各个学习结果进行整合从而获得比单个学习器更好的学习效果的一种机器学习方法。
- `bootstrapping`：在大小为S的数据集上进行S次有放回的抽样，建立新的训练集。由于会有很多重复出现的样本，新的训练集中约包含原有数据集63.2%的数据。
- `Bagging`：通过bootstrapping多次在原有数据集中抽样生成n个新的数据集，然后使用同一种弱学习器分别进行n次学习得到n个学习器，由这n个学习器在测试集上进行预测，最终结果为这n个学习器预测结果的平均值。
- `AdaBoosting`：每次使用全部的样本，每轮训练改变样本的权重。下一轮训练的目标是找到一个函数f 来拟合上一轮的残差。当残差足够小或者达到设置的最大迭代次数则停止。Boosting会减小在上一轮训练正确的样本的权重，增大错误样本的权重。

## 二、实验任务

---

- 基础任务
  - 使用两种集成学习方法（`AdaBoosting`、`Bagging`）和两种分类器（SVM和决策树）完成实验。
- 拓展任务
  - 使用其他分类器。
  - 分析不同feature的效果。
  - 调整集成学习算法的参数，分析他们对结果的影响。
  - 使用其他方法来达到更好的效果。

## 三、实验设计

---

### Feature 选取和处理

本次实验feature一共有 `reviewerID`, `asin`, `reviewText`, `overall`, `votes_all`, `votes_up`，它们的具体含义为：

- `reviewerID`：评论者ID。
- `asin`：物品ID。
- `reviewText`：评论文字。
- `overall`：评分。
- `votes_all`：这篇评论收到的投票总数。
- `votes_up`：收到的投票中赞同的总数。

要在训练中让他们起到作用，需要对他们进行进一步处理。

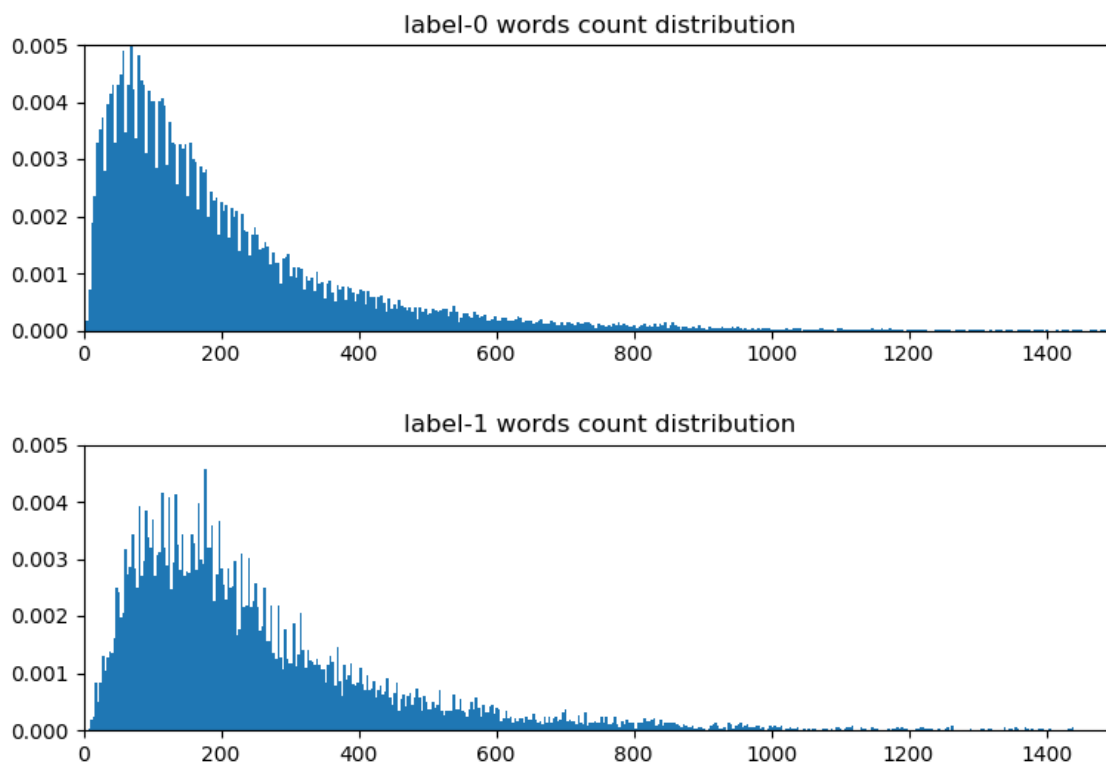
#### 1. `reviewText`

评论文本是分类的一个重要依据，可以按评论文本的以下三个特性作为feature进行训练：

- 文本单词数：文本包含英文单词的个数。
- 文本长度：`len(reviewText)`。
- 文本词向量：文本中包含不同的词，可以将文本转换为词向量作为训练数据进行训练。

现对各个特性在两个label情况下的差异性进行探究，以选取差异最大的特性来训练。

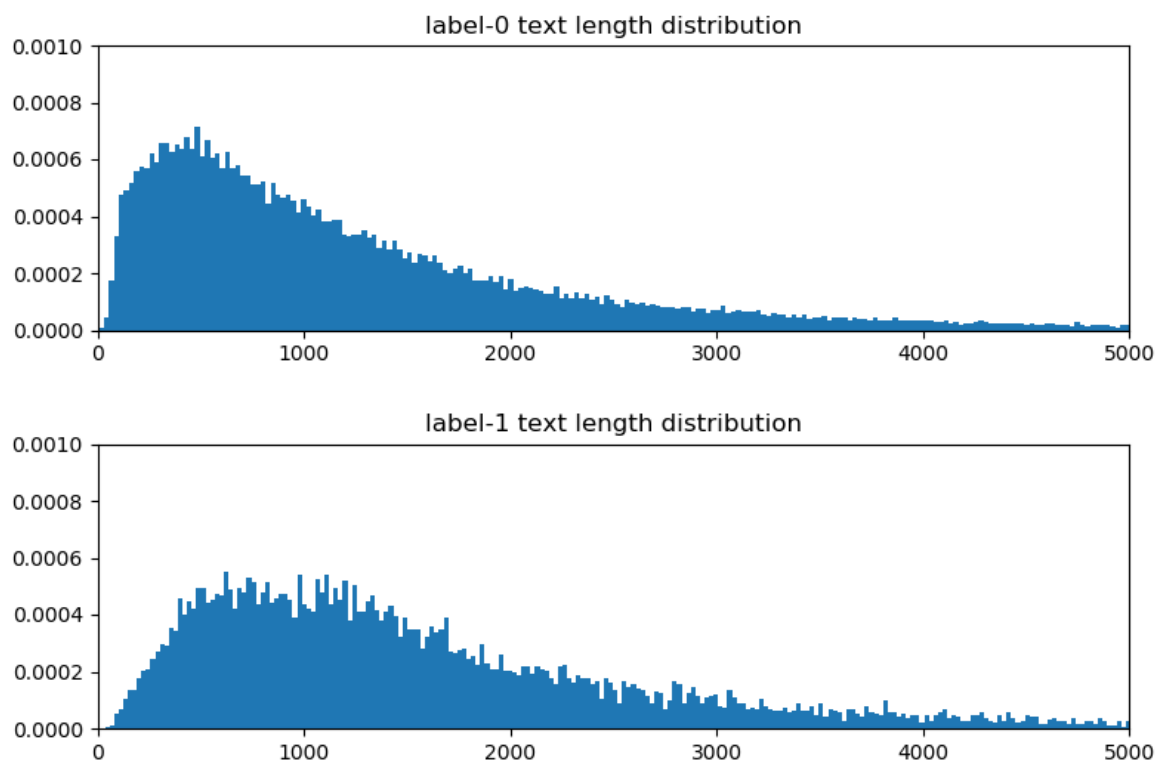
### 文本单词数



上图中横轴为词数量，纵轴为概率。从图中可以看出，两种文本词数量都集中在0-600之间，label为0的文本词数量为75处达到峰值且分布较为集中，label为1的文本词数量在词数量为190处达到峰值，分布较为分散。

这可以作为一个feature，但从图中看出两种文本的词数量差异并不显著，因此该feature作用不会太大。

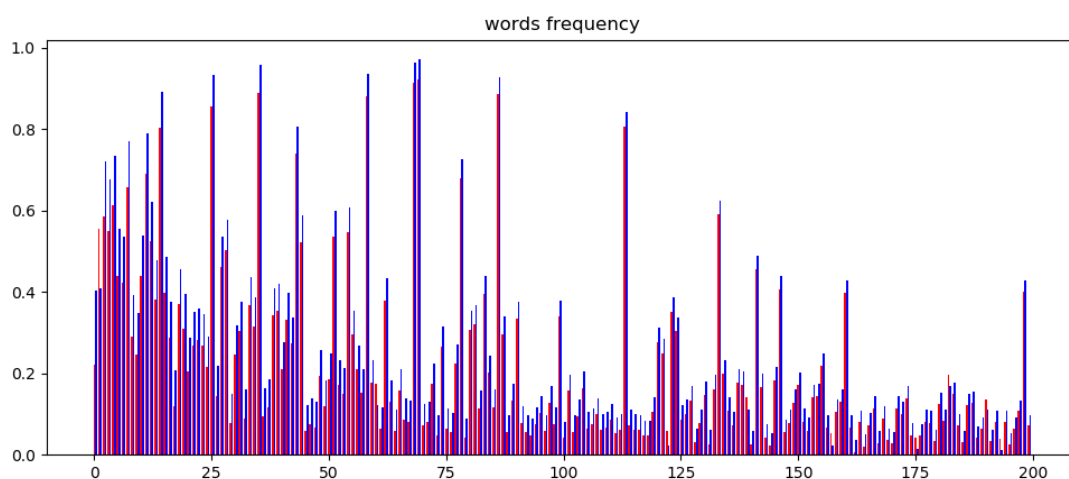
### 文本长度



文本长度特征和文本单词数分布情况相似，两中文本具有分布差异但同样不够显著。

## 文本词向量

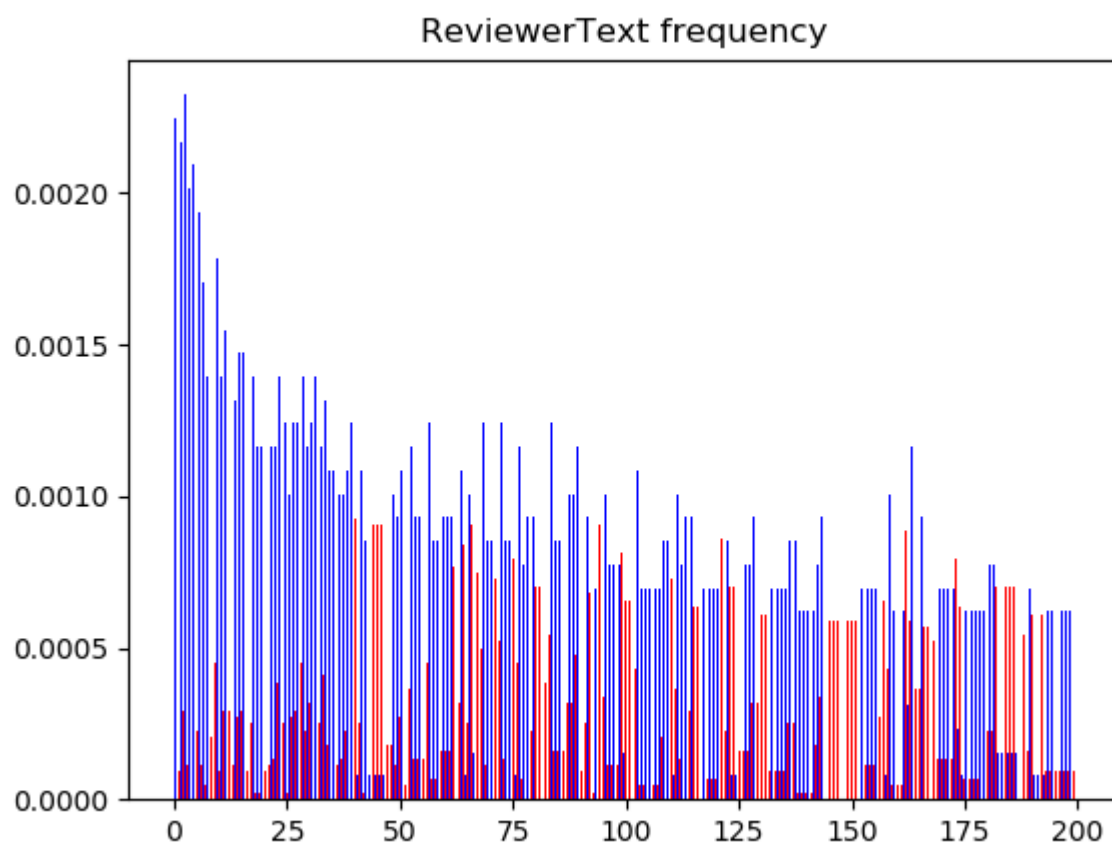
记 $P(w_i|label_i)$ 为单词 $w_i$ 在label为 $label_i$ 的 `reviewText` 中出现的概率。下图为 $||P(w_i|0) - P(w_i|1)||$ 最大的前200词的在两种编辑文本中出现的词频对比图。其中，红色为在label为1中的词频，蓝色为label为0中的词频。



可以看出，两者词频有差距且差距比较明显，可以根据词向量来训练弱学习器。

## 2. `reviewerID`

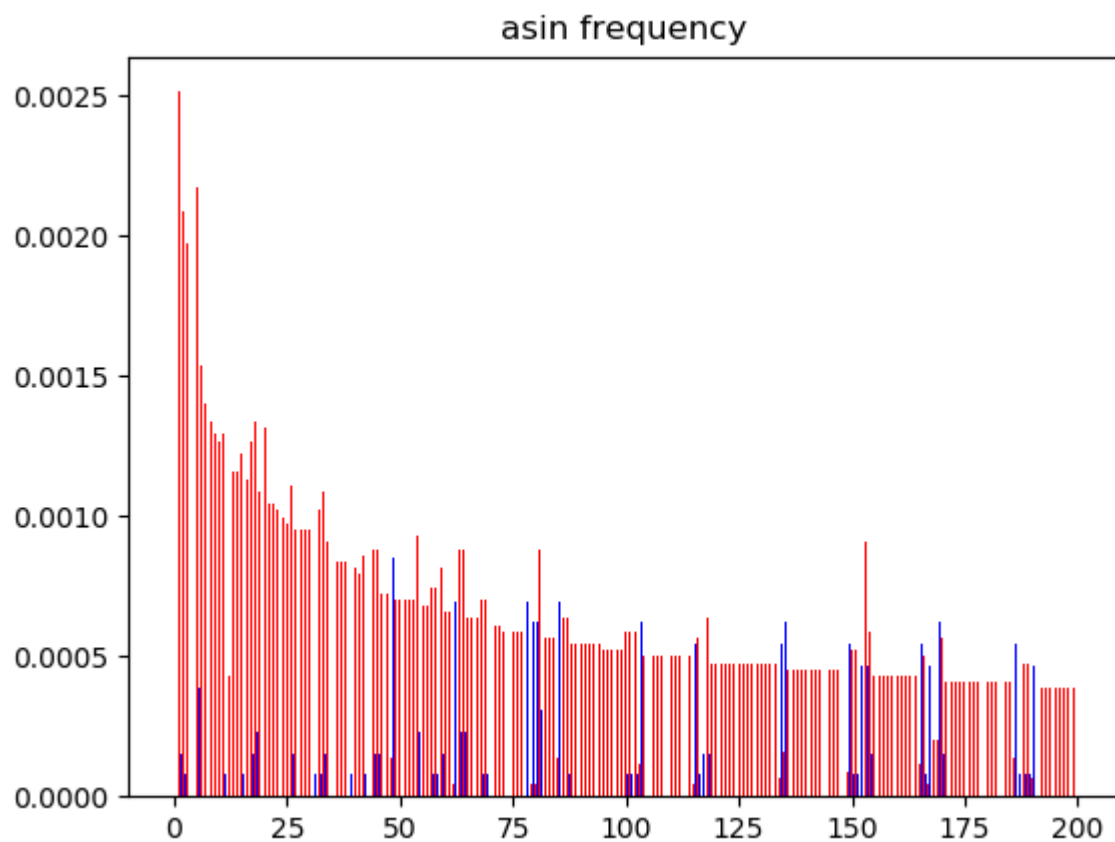
根据常识判断，每个人对商品的评价会偏向于高质量或者低质量两个方面，因此 `reviewerID` 是分类的重要依据。实际上,  $reviewerID_i$  在两类中出现概率之差最大的200个分布如下（红色是  $reviewerID_i$  出现在label为1中的概率，蓝色是出现在label为0中的概率）：



因此，`reviewerID` 是一个非常重要的判据。但是，`reviewerID` 作为一串无规则数字，使用SVM分类比较困难，但是使用 `DecisionTree` 和贝叶斯分类很容易。

### 3. `asin`

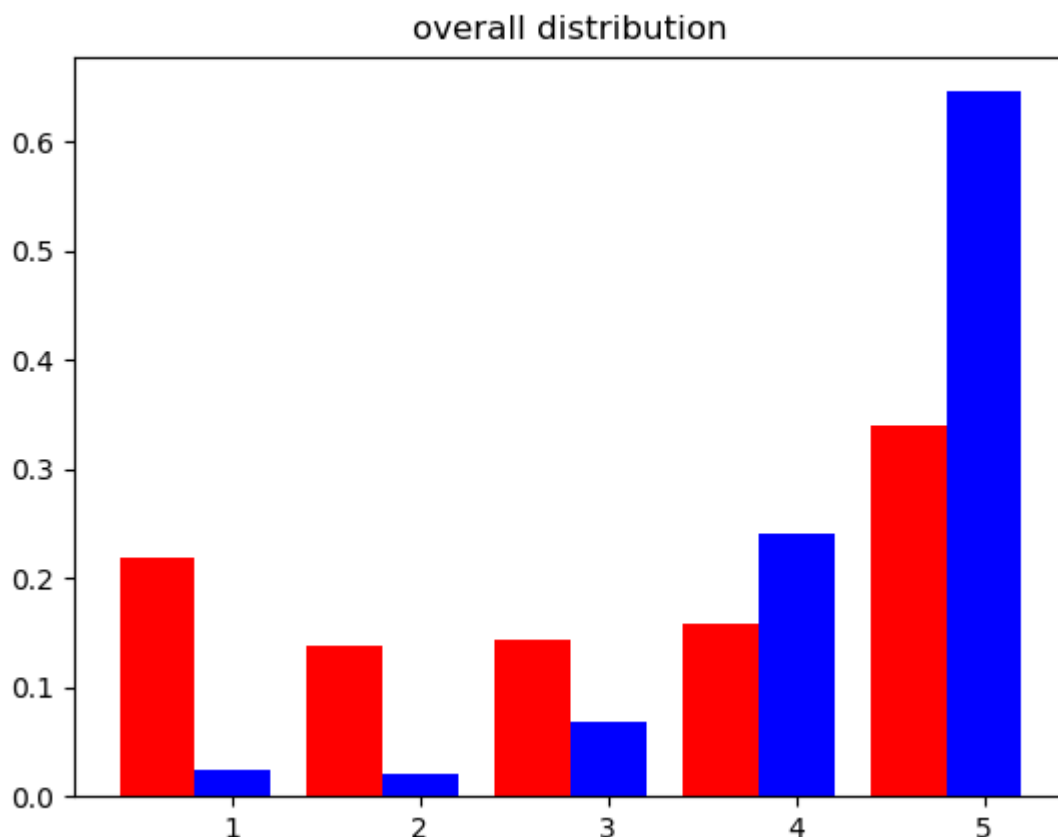
对于商品而言，好的商品有更多的高质量评论，因此 `asin` 应该也会是一个重要的分类依据，实际上,  $asin_i$  在两类中出现概率之差最大的200个分布如下（红色是  $asin_i$  出现在label为1中的概率，蓝色是出现在label为0中的概率）：



因此，`asin`是一个非常重要的判据。但是，`asin`作为一串无规则数字，使用SVM分类比较困难，但是使用`DecisionTree`和贝叶斯分类很容易。

#### 4. `overall`

高质量评论的评分实际分布如下（其中，红色是在label为0的评论中出现的概率，蓝色是在label为1中出现的概率）：



很明显，低质量评论评分分布较为随机，而高质量评分分布大多集中在高分。

## 5. `votes_all` 和 `votes_up`

由于这两个feature只有在训练集中才有，所以只能作为训练集的权重来发挥作用。因此，在训练的过程中，可以采用 `votes_up/votes_all` 来做权重。

## 最终feature选取

在实际训练时，根据学习/预测速度、实际效果、参数格式三个方面因素考虑，最终在不同学习器下，得出不同的训练集。

- DecisionTree: `reviewText`长度, `reviewID`, `asin`, `overall`
- SVM: 使用tfidf学习reviewText并将其转换为词向量矩阵，再拼接上overall转换为压缩矩阵后作为训练数据。

## 集成学习参数设置

### 初始权重

初始权重设置有三种选择:

- 所有训练样本相等权重
- 使用`votes_up/votes_all`作为权重
- 在学习器训练时添加 `class_weights=balanced` 参数，以平衡label为1和0训练样本数量带来的偏差。

开始的时候由于训练参数设置有些问题，不小心提交了一个全0的结果上去，得到了50%的score，让我一度以为测试集中两类评论的数量是五五开的，让我在很长一段时间里都不敢提交得到的结果中0和1数量相差太大的结果。到了最后才发现好像实际测试集并不是这样.....也就是说 `class_weights=balanced` 不应该添加。我之前还想来很多办法来平衡两种label的数量差对训练结果的影响，比如说在ADABOOST中每一轮都分别对label为0和label为1的样本权重归一化，再最终归一化等。最后才发现原来训练集和测试集label分布应该是有相同规律的。

## 迭代轮数

为了避免最终出现两个label分类概率相等的情况，迭代轮数最好为奇数。

### Bagging

对于bagging，只要尽量让每一次的训练集之并集覆盖到全数据集即可，由于每一次选取的训练集中有36.8%的数据不在训练集之中，只要经过11轮迭代之后，数据集中没有被学习的数据占比就可以下降到0.00167%。

### AdaBoost.M1

对AdaBoost.M1采用SVM和对应feature，在自己划分的测试集上得出结果如下：

| 轮数   | 1     | 5     | 11    | 15    | 21    |
|------|-------|-------|-------|-------|-------|
| 正确率% | 78.59 | 78.21 | 78.20 | 78.21 | 78.21 |

正确率随着测试轮数增加而减小，猜测原因是测试过程中噪声样本权重不断增加，导致学习了过多噪声，要解决这个问题，可以在固定轮数后清理噪声样本。

对AdaBoost.M1采用DecisionTree和对应feature，在自己划分的测试集上得出结果如下：

| 轮数   | 1     | 5     | 11    | 15    | 21    |
|------|-------|-------|-------|-------|-------|
| 正确率% | 73.76 | 72.89 | 74.01 | 73.79 | 74.73 |

所以，对于DecisionTree+AdaBoost.M1，11轮比较合适。

## 弱学习器参数设置

### DecisionTree

经过尝试以后采用了如下参数设置：

```
1 | clf = DecisionTreeClassifier(min_samples_split=30)
```

### SVM

尝试了sklearn中的多种SVC及其核函数，最终在和柳瑞阳同学交流后使用了这样的线性SVM：

```
1 | clf = LinearSVC()
2 | clf = CalibratedClassifierCV(clf, method='sigmoid', cv = 3)
```

## 贝叶斯

采用贝叶斯学习器作为集成学习中的弱学习器，训练结果全0.....

## 三、实验结果及分析



### 实验结果

以下实验结果具体数据均在output文件夹中。

|             | SVM    | DecisionTree |
|-------------|--------|--------------|
| Bagging     | 80.523 | 72.376       |
| AdaBoost.M1 | 80.230 | 73.274       |

### Kaggle截图

最好结果和排名：

|  |                   |  |         |    |    |
|--|-------------------|--|---------|----|----|
| 13   | vegetable is good |  | 0.80523 | 23 | 2h |
| <b>Your Best Entry</b>  |                   |  |         |    |    |
| Your submission scored 0.73274, which is not an improvement of your best score. Keep trying!             |                   |  |         |    |    |

四种不同组合的结果：

| Submission and Description  | Public Score | Use for Final Score                 |
|---|--------------|-------------------------------------|
| <b>BAGGING_DT.csv</b><br>2 hours ago by sjy2016011395<br>BAGGING+dt   | 0.73274      | <input type="checkbox"/>            |
| <b>ADABOOST_DT.csv</b><br>2 hours ago by sjy2016011395<br>adaboost+DT | 0.72376      | <input type="checkbox"/>            |
| <b>BAGGING_SVM.csv</b><br>a day ago by sjy2016011395<br>BAGGING+SVM   | 0.80523      | <input checked="" type="checkbox"/> |
| <b>ADABOOST_SVM.csv</b><br>a day ago by sjy2016011395<br>ADABOOST+SVM | 0.80230      | <input type="checkbox"/>            |

### 结果分析

从结果可以看出，采用SVM的Bagging结果最好。使用DecisionTree的Bagging效果最差。

- SVM整体优于DecisionTree
  - 训练数据差异：造成这个结果的原因和两个弱学习器训练数据的差异有关。因为DecisionTree在词向量处理过程上速度太慢，于是没有使用词向量，而是只使用了文本长度和其他几个feature作为训练数据来训练，这造成了信息丢失。



- SVM使用Bagging优于使用AdaBoost.M1
  - AdaBoost.M1在训练时会更多的权重放在出错的样本上，这也决定了AdaBoost.M1在训练时受噪声影响更大。为了避免受到噪声影响，需要定期清理权重过高样本或者将权重过高样本权重调低。
- DecisionTree使用AdaBoost.M1优于Bagging
  - 这一点的主要原因还是因为DecisionTree的训练样本噪声没有SVM的训练样本噪声大。即主要还是因为我在使用两种学习器的时候使用的数据不一样。

## 四、实验总结

这次实验是真的磨人...从弱学习器的参数设置到集成学习学习器的设置，从feature的选取，还有词向量几种转换方式的设置，都需要多次实验，但是实际提交次数只有每天三次，要看到最后结果还是很难，数据集中划分的测试集上的测试结果实际上和真正测试集上测试的结果差距还比较大。

总的来说，这次实验让我体会到了使用机器学习方法来完全从头来处理一个feature不是特别明显的分类问题的绝望。最后的结果应该并不是最优结果，因为要得到最优结果需要考虑的参数太多了，这个实验花了太多时间，只能这样提交了。

谢谢老师、助教、同学在实验中提供的帮助。

## 五、代码说明

`util.py` 是我用来做feature分析的一些代码，可以忽略。

`ensemble.py` 是主要程序，运行时，在src文件夹下执行 `python ebsemble.py`，若需要修改集成学习参数或者学习器等，在文件中，修改以下参数：

```
1  BAGGING_T = 11                #BAGGING迭代次数
2  ADABOOST_M1_T = 11            #ADABOOST.M1迭代次数
3  TEST_ON_TRAIN_SET = True      #是否在训练集中划分测试集
4  CLASSIFIER_TO_USE = SVM       #使用何种弱学习器
5  ENSEMBLE_WAY = ADABOOST_M1   #使用何种集成学习方法
6  SAMPLE_WEIGHT = EQUAL_WEIGHT #样本权重，为EQUAL_WEIGHT则为相同权重，否则为
   votes_up/votes_all
7  USE_TEXT_VECTOR = True        #是否使用词向量
```

执行完毕后，结果会输出在output/目录下的result.csv中。