Ensemble-Learning 实验报告

计 74 顾掀宇 2017011401

2020 年 5 月 8 日

1 实验目标

在给定的 Amazon 评论-评分数据集上，基于两种基本的分类算法 SVM 和 Decision Tree，实现两种集成学习算法 Bagging 和 AdaBoost.M1，通过不同的特征提取方式, 以实现对于评论进行评分预测，用 RMSE 来评估预测能力。

2 实验设计

整个实验框架进行了工程化，主要分为如下模块：

- main.py，暴露给用户的执行入口
- word2vec.py，用于实验数据的预处理，生成词向量与数据集
- dataset.py，划分并生成训练集、验证集
- settings.py，框架用到的配置类与公用函数
- predict.py，用于 validation 以及生成 test
- train.py，训练函数，根据不同的配置选择调用不同的训练函数
- svm.py，关于 svm 的训练与预测函数调用
- decision_tree.py，关于决策树的训练与函数调用

实验运行环境：

- scikit-learn == 0.21.3
- gensim == 3.8.3
- joblib == 0.13.2
- numpy, pandas...

2.1 数据集划分

将 train.csv 内的数据按 9:1 的比例划分为测试集与验证集（毕竟测试集完全黑盒没有标签），通过 `re.split(r'\W', string)`对于评论进行分词，并筛除掉部分停用词，通过下一步的特征提取，通过 numpy 存储转换过后的数据集。

2.2 特征提取

2.2.1 词向量模型

词向量模型，也就是将单词向量化，将单词用一个特定维数的向量来表示。主流的词向量表示模型分为两种：

- one-hot representation，向量的分类只有一个 1，其它全为 0，1 的位置对应词在词典中的索引。表示方便，如果加和起来可以很容易的表示一个文档向量。但是一个词向量所能传达出的信息有限，比如不能通过距离表现两个词之间的关系，维数是词典长度，所占空间过多。
- distributed representation，所有的词向量构成了一个词向量空间，每个词占据空间上的一个点，可以更好的表示每个词之间的关系。

这里我是用了 gensim 库提供的 word2vec 完成对于词向量的构建。显然语料的规模大小会对词向量结果产生显著影响，我分别尝试了自行对于语料库进行训练以及通过已有的大规模英文语料库进行训练。

我们需要对于一条评论转化为特征向量，我们现在拥有了每个词向量，那么应该如何将多个词向量结合转化为一篇文章的向量呢：

- 将一句话中的所有词向量进行平均，得到了平均词向量作为一篇文章的向量。这种方法方便但不是一种好的做法，没有考虑到单词的顺序，也没有考虑词的重要性的区别。
- TFIDF-weighting word vectors，对句子中所有的词向量根据 TFIDF 权重加权求和，句子中更重要的词所占比重更大，缺点也是没有考虑到单词的顺序。
- 使用 doc2vec 技术
- ...

2.2.2 TF-IDF 加权词向量

$TFIDF = TF \cdot IDF$，TF 即 Term Frequency，表示词频，IDF 为 Inverse Document Frequency。TF 表示词条在所有文件中出现的次数。当包含该词的文档越少，IDF 越大，说明这个词条具有很好的类别区分能力。在这里我选择使用 TF-IDF 对于词向量进行加权。使用 sklearn 提供的函数获取训练集的训练集特征。

2.2.3 几种特征提取方法的结果对比

这里简单的对于几种特征选取方法进行对比，分类 + 集成算法是 AdaBoost.M1 + 决策树，理论上我认为 TF-IDF 加权后效果应该更好一些，但是并没有，猜测 Word2Vec 从某种程度上已经考虑了 TF-IDF 的因素，因此效果不及单纯地平均；此外我下载的外部语料库可能规模不及 train.csv，效果反而出现了倒退。

除此之外，这里我手动训练的词向量大小为 128 维，事实上我也尝试了 256 维的词向量，表现有略微的提升。

词向量选取方法	RMSE
手动训练 + Avg	1.03932
外部语料库训练 + Avg	1.11606
外部语料库训练 + TF-IDF Avg	1.08040

2.3 底层分类器

2.3.1 SVM

通常传统意义上的 SVM 用于二分类问题，为了实现多分类，主要有以下两种方式：

- one-vs-all，训练时依次把某个类别的样本归为一类，其它剩余样本归为另一类，k 个类别的样本构造出了 k 个 SVM。
- one-vs-one, 任意两类样本之间设计一个 SVM，k 个类别的样本需要设计 k(k-1)/2 个 SVM。当对一个未知样本进行分类时，最后得票最多的类别即为该未知样本的类别。

这里我使用了 sklearn 提供的 LinearSVC 函数，根据其官网介绍，面对多分类时，使用 one-vs-all 的策略。

2.3.2 决策树

决策树的训练使用 sklearn.tree 即可。

2.4 集成学习算法

2.4.1 Bagging 算法

Bagging 即 Bootstrap aggregating，算法如下：

根据实验证明，Bagging 更适合于那么不稳定的分类器，比如 Decision Tree 就是一个不稳定的分类器，在决策树上 Bagging 提升的效果更加显著。此外，理论上 Bagging 可以并行训练，训练速度更快。Bagging 可以降低方差，但是不可以降低 bias。

2.4.2 AdaBoost.M1

算法如下：

Algorithm 1 Bagging

Require:

大小为 m 的训练集

1: 从给定的 m 大小的训练集里有放回的采样出大小为 m 的训练集

2: 通过采样出的训练集训练出一个分类器。

3: **return** 通过投票取众数决定标签。(这里由于是评分，我们可以对其取平均数来计算)。

Algorithm 2 AdaBoost.M1

Require:

初始化权重为 1/N，N 为测试集大小

dst 到 mid 的变换矩阵 H2

1: 参考课堂讲义所给的方案，训练算法难以为每个样例指定权重，我们可以使用 resample 的策略，根据测试集每个样例的权重，使用 np.random.choice 生成一个样例集合。

2: 训练底层的分类器，根据权重计算出 ε_t，即加权 error rate

3: 如果 $\varepsilon_t > 0.5$，结束训练

4: $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$

5: 对于正确分类的，$\omega_{new} = \omega_{old} \cdot \beta_t$；错误分类的，权值保持不变。

6: 权值标准化（使得和为 1）

7: **return** 对于每个分类器，用 $log[1/\beta_t]$ 加权结果;

3 实验结果

最好的结果为 DTREE + Bagging，最终 Kaggle 上显示的 RMSE 为 0.98878（而且事实证明我添加的 Feature 好像都是副作用，比如加入 TF-IDF 加权以及外部语料库)，以下的 RMSE 取自验证集（和 Kaggle 上的测试集几乎没有太大差别)。

Ensemble + Classifier	RMSE
SVM	1.335426387474666
BAGGING(20) + SVM	1.2953563918158515
ADA + SVM	1.326062351494287
DTREE	1.4288106942488918
BAGGING(20) + DTREE	1.0008095586710155
BAGGING(30) + DTREE	0.9832412544397134
BAGGING(50) + DTREE	0.9796404999237777
ADA + DTREE	1.0363696812989727

写完的时候排在第二名，因为也就三个人写了... 截图如下，用户名 XianyuGu：

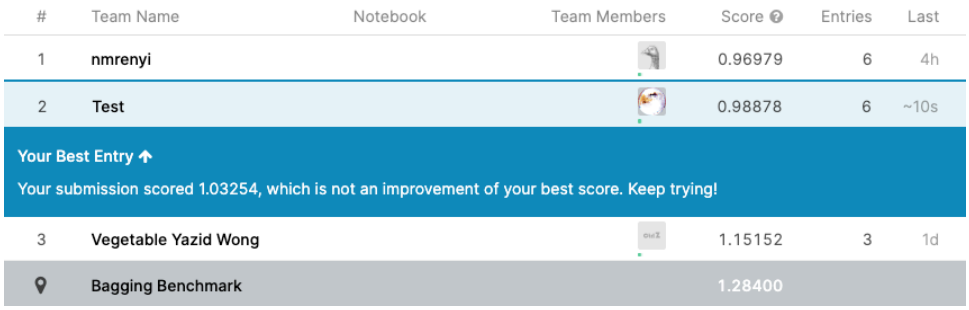

图 1: 排行榜

4 实验分析

1. 对于 SVM，我们可以看到使用 BAGGING 亦或是 AdaBoost.M1 都没有产生明显的增强效果，因为 SVM 是稳定的算法，在集成学习算法中都没有得到太多增益。

2. DTREE 模型稳定性较差，在集成学习算法中得到了明显的增益效果。尤其是在增加 BAGGING 的数目之后表现会得到进一步提升，这也确实说明了 DTREE 是不稳定的算法。

3. 尽管 Bagging 只能降低方差不能降低 Bias，但是在本次实验中，AdaBoost.M1 的效果不及 Bagging，具体原因下一节分析。

4. Bagging 确实利于并行化操作，而且操作简单，同时 Boosting 需要存储计算多个 weights，可能速度也确实比较慢。

4.1 关于 AdaBoost.M1 的进一步分析

实验结果确实证实了这样一点：Boosting might hurt performance on noisy datasets，对于我们的五分类任务而言，用准确率来评估得分这件事情确实比较困难（如果是 5 分打了 4 分其实也没什么大问题），因此用错误率来评估退出 AdaBoost 的计算比较严苛（对比而言，RMSE 是一个更加客观的衡量标准）。所以这里我们不妨可以对于 ε_t 降低标准，等待 $\varepsilon_t > 0.6$ 时再跳出循环。除此之外，AdaBoost 越往后，一些容易被误分类的样本所占比例更高，分类器在验证集的表现可能会越差，我们需要对于其所占权重需要更加“宽容”一些，因为毕竟它面对的是那些难啃的数据。

修改了最后的权重函数曲线，横轴是错误率，纵轴是通过 log 算出来的 weight。绿色的修改过后的，红色的是原来的。可以看到，错误率高的分类器，所占权重也会比原来大一些。

简单对比一下初始版本的和修改过后的，大概性能有略微的提升，但好像没有太多变化，不过我仍然相信问题的根源就在 Accuracy 这一评价指标用在五分类问题上不太适用。

Ensemble + Classifier	RMSE
ADA + DTREE	1.03932
ADA + DTREE V2	1.03089

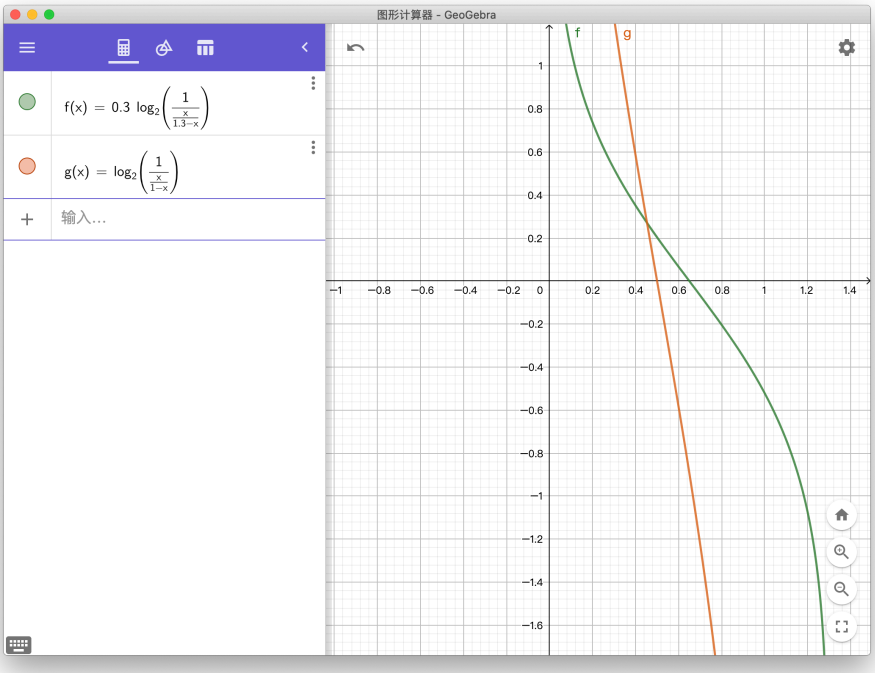

图 2: 最后投票加权的函数曲线

5 实验小结

这次实验，我手动实现了两种集成学习算法，对于里面内在的原理进行了分析，也手动尝试了其它的 Feature，同时也第一次体验了 Kaggle 的玩法，还算很有收获，感谢助教评阅！