

机器学习概论 实验二

计52 路橙 2015010137

一、实验描述

实现**ensemble learning algorithms**，对50,000个淘宝商品评论进行学习并可以对淘宝评论进行情感分析的三分类（positive: 1, neutral: 0, negative: -1）。

分别实现两种基础的分类算法：**SVM** and **Decision Tree**，分别实现两种集成学习算法**Bagging** and **AdaBoost.M1**，两两结合共4种分类方法，来对文本数据进行分类。

特征选择及提取需要自行设计实现。

二、实验设计

实验分为如下子模块：

- 数据集划分。用于训练及本地测试
- 特征提取。将原始文本数据提取出适合机器学习的特征向量，用于训练及测试。
- 训练模型，包括分类算法以及集成学习算法。
- 预测，选择与分类算法相适配的算法。

具体而言，如下介绍各个模块：

2.1. 数据集划分及数据预处理

本部分对应代码 `dataset.py`。

2.1.1. 数据预处理

由于所给训练数据中包含标点符号和一些对于语义理解无关的词语，因此需要对数据进行进一步的预处理。

在 `filter_words()` 中：

- 去除所有的标点符号和换行符（不包括空格）
- 根据空格将句子划分为词列表
- 去除列表中的空元素
- 去除列表中的中文停用词
- 频次截断

通过以上步骤，可以将训练数据和测试数据更关注于句子传达出的情感。

停用词

停用词是指在信息检索中，为节省存储空间和提高搜索效率，在处理自然语言数据（或文本）之前或之后会自动过滤掉某些字或词，这些字或词即被称为Stop Words（停用词）。这些停用词都是人工输入、非自动化生成的，生成后的停用词会形成一个停用词表。但是，并没有一个明确的停用词表能够适用于所有的工具。甚至有一些工具是明确地避免使用停用词来支持短语搜索的。

停用词往往是那些信息含量很低的无用词汇。这些词汇往往出现频率很高，如“的”、“这样”、“这”、“不过”、“哈哈”等，这些词语往往在特征向量中是主要的噪声，需要剔除。

停用词表取自github中[暨南大学的一个项目](#)，共有1208个常用的中文停用词。

2.1.2. 数据集划分

在 `divide_set()` 中，调用 `sklearn.model_selection.train_test_split` 函数进行训练集与验证集的划分。

此处，我采用 9:1 的比例划分了训练集与验证集。

而最终的文本数据的转化在特征提取中实现。此处需要调用特征提取函数，将原始数据提取特征后保存为 `npz` 文件，方便之后训练时的直接读取。

2.1.3. 数据读入

在做好以上两步后，每次训练或测试时，只需要调用对应的 `load` 函数，将 `npz` 文件导入到内存中即可使用。

这样的设计极大地节省了训练和测试的数据预处理成本。

2.2. 文本特征提取

本部分对应代码 `wordvec.py` 和 `tfidf.py`。

本次实验中，我共实现了两种特征提取算法，**词向量模型**与**tf-idf (term frequency-inverse document frequency)** 方法。

2.2.1. word2vec语言模型

2.2.1.1. 单词的向量化表示

所谓的word vector，就是指将单词向量化，将某个单词用特定的向量来表示。将单词转化成对应的向量以后，就可以将其应用于各种机器学习的算法中去。一般来讲，词向量主要有两种形式，分别是稀疏向量和密集向量。

所谓**稀疏向量**，又称为**one-hot representation**，就是用一个很长的向量来表示一个词，向量的长度为词典的大小N，向量的分量只有一个1，其他全为0，1的位置对应该词在词典中的索引。这种表示方法不需要繁琐的计算，简单易得，但是缺点也不少，比如长度过长（这会引发维数灾难），以及无法体现出近义词之间的关系。但是，用这种稀疏向量求和来表示文档向量效果还不错，清华的长文本分类工具 `THUCTC` 使用的就是此种表示方法。

至于**密集向量**，又称**distributed representation**，即分布式表示。最早由Hinton提出，可以克服one-hot representation的上述缺点，基本思路是通过训练将每个词映射成一个固定长度的短向量，所有这些向量就构成一个词向量空间，每一个向量视为该空间上的一个点。此时向量长度可以自由选择，与词典规模无关。这是非常大的优势。这种表示方式更精准的表现出近义词之间的关系，比之稀疏向量优势很明显。

word2vec的原理大致来说，就是构建了一个多层神经网络，然后在给定文本中获取对应的输入和输出，在训练过程中不断修正神经网络中的参数，最后得到词向量。

2.2.1.2. word2vec的使用

所谓的语言模型，就是指对自然语言进行假设和建模，使得能够用计算机能够理解的方式来表达自然语言。**word2vec**采用的是**n元语法模型**(n-gram model)，即假设一个词只与周围n个词有关，而与文本中的其他词无关。

而在本次实验中，我使用 `gensim` 库进行了词向量的构建。具体而言，尝试了两种方法：

- 通过训练集和测试集的词语直接训练词向量。

- 使用已有的中文词向量模型。

分别介绍如下。

2.2.1.3. 自行训练词向量模型

由于训练集和测试集在预处理后已经可以直接拿来训练词向量，因此，我尝试了自行训练词向量的方法。

具体而言，大致过程为：

1. 构建词向量模型。

```
imdb_w2v = Word2Vec(size=n_dim, min_count=10)
```

2. 建立词汇表。

```
all_text = x_train + x_validation + x_test
imdb_w2v.build_vocab(all_text)
```

3. 训练词向量模型。

```
imdb_w2v.train(all_text, total_examples=imdb_w2v.corpus_count, epochs=imdb_w2v.epochs)
```

4. 保存词向量模型。

```
imdb_w2v.save('model/w2v_model.pkl')
```

经过如上的过程，我们可以利用已知的数据训练好一个词向量模型。在我的方法中，选取的词向量维数为 `n_dim=128`。

而我们最终需要分析的是句子，在句子已经分词后，得到每一个词向量，如何将它们结合呢？一般而言，有如下三种常用的方法：

- 将一句话中所有词的词向量进行平均，得到平均词向量作为该句子的向量。这种方法没有考虑词之间的关系，也没有考虑词的重要性区别。此外，这种方法需要一句话中含有的词个数较少，当文本长度太长时，这种方法的表示会趋于发散。
- 使用 `doc2vec` 技术。这种方法一般而言需要保证一句话中含有的词的个数较多，否则效果不明显。
- 将词向量接入时序模型中，如RNN或LSTM。

由于本次实验我们句子的长度较短，且不允许使用高级模型，因此我最终选择了第一种方法来表示一句话。对应的代码为 `buildWordVector` 函数。

2.2.1.3. 使用已有的词向量模型

由于自行训练的词向量模型限于数据量大小，有可能无法精准表示词语的含义，因此我下载了[网上某个博客提供的词向量模型](#)，并将该词向量模型用于特征的提取。

值得一提的是，网上的词向量模型基本都是基于新闻或百科词条进行训练，得到的词向量偏书面化。但本次实验中我们用的语料偏口语化，因此使用现有的词向量模型会有一定的偏差。

2.2.2. TF-IDF方法

2.2.2.1. TF-IDF概念

TF-IDF (term frequency-inverse document frequency) 是一种用于资讯检索与资讯探勘的常用加权技术。**TF-IDF**是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。TF-IDF加权的各种形式常被搜寻引擎应用，作为文件与用户查询之间相关程度的度量或评级。除了TF-IDF以外，因特网上的搜寻引擎还会使用基于连结分析的评级方法，以确定文件在搜寻结果中出现的顺序。

2.2.2.2. 词频

在一份给定的文件里，**词频 (term frequency, TF)** 指的是某一个给定的词语在该文件中出现的次数。这个数字通常会被归一化（分子一般小于分母 区别于IDF），以防止它偏向长的文件。（同一个词语在长文件里可能会比短文件有更高的词频，而不管该词语重要与否。）

2.2.2.3. 逆向文件频率

逆向文件频率 (inverse document frequency, IDF) 是一个词语普遍重要性的度量。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到。

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

2.2.2.4. 主要原理

TFIDF的主要思想：如果某个词或短语在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。TFIDF实际上是：TF * IDF，TF词频(Term Frequency)，IDF反文档频率(Inverse Document Frequency)。**TF**表示词条在文档d中出现的频率（另一说：**TF词频(Term Frequency)**指的是某一个给定的词语在该文件中出现的次数）。IDF的主要思想是：如果包含词条t的文档越少，也就是n越小，IDF越大（见后续公式），则说明词条t具有很好的类别区分能力。如果某一类文档C中包含词条t的文档数为m，而其它类包含t的文档总数为k，显然所有包含t的文档数n=m+k，当m大的时候，n也大，按照IDF公式得到的IDF的值会小，就说明该词条t类别区分能力不强。（另一说：**IDF反文档频率(Inverse Document Frequency)**是指果包含词条的文档越少，IDF越大，则说明词条具有很好的类别区分能力。）但是实际上，有时候，如果一个词条在一个类的文档中频繁出现，则说明该词条能够很好代表这个类的文本的特征，这样的词条应该给它们赋予较高的权重，并选来作为该类文本的特征词以区别与其它类文档。这就是IDF的不足之处。

2.2.2.5. 提取方法

使用 `sklearn.feature_extraction.text` 库的 `CountVectorizer`, `TfidfTransformer` 方法来提取TF-IDF特征。

需要注意的是：

- 由于训练集和测试集分开提取特征会导致两者的特征空间不同，比如训练集里某个单词在测试集里序号很有可能就不同了，或者根本就不存在在测试集里。所以这里先用所有文档共同提取特征（`counts_v0`），然后利用得到的词典（`counts_v0.vocabulary_`）再分别给训练集和测试集提取特征。然后开始训练与测试。
- 使用 `TfidfTransformer` 得到的特征是一个很大的稀疏矩阵，需要调用 `toarray()` 方法来获得numpy数组。

2.3. 训练模型

2.3.1. 分类器

分类器主要为SVM和Decision-Tree。由于我的方法使用了两种不同的特征，对于每一种特征，分类器的选择与表现也不尽相同。

2.3.1.1. SVM (对于分类问题，为SVC)

SVC (C-Support Vector Classification)：支持向量分类，数据拟合的时间复杂度是数据样本的二次方，这使得算法很难扩展到10000个数据集。由于SVM最初为了二分类问题实现，但本次实验需要的是三分类问题。对于SVM多分类问题，往往有以下方法：

SVM解决多分类问题的方法

SVM算法最初是为二值分类问题设计的，当处理多类问题时，就需要构造合适的多类分类器。目前，构造SVM多类分类器的方法主要有两类：一类是直接法，直接在目标函数上进行修改，将多个分类面的参数求解合并到一个最优化问题中，通过求解该最优化问题“一次性”实现多类分类。这种方法看似简单，但其计算复杂度比较高，实现起来比较困难，只适用于小型问题中；另一类是间接法，主要是通过组合多个二分类器来实现多分类器的构造，常见的方法有one-against-one和one-against-all两种。

1. 一对多法 (one-versus-rest,简称1-v-r SVMs)。训练时依次把某个类别的样本归为一类,其他剩余的样本归为另一类，这样k个类别的样本就构造出了k个SVM。分类时将未知样本分类为具有最大分类函数值的那类。
2. 一对一法 (one-versus-one,简称1-v-1 SVMs)。其做法是在任意两类样本之间设计一个SVM，因此k个类别的样本就需要设计 $k(k-1)/2$ 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。Libsvm中的多类分类就是根据这个方法实现的。
3. 层次支持向量机 (H-SVMs)。层次分类法首先将所有类别分成两个子类，再将子类进一步划分成两个次级子类，如此循环，直到得到一个单独的类别为止。
4. 其他多类分类方法。除了以上几种方法外，还有有向无环图SVM (Directed Acyclic Graph SVMs，简称DAG-SVMs) 和对类别进行二进制编码的纠错编码SVMs。

本次实验中，我分别尝试了 `ovr` 与 `ovo` 方法，得到的结果不尽相同。

不同特征对SVM的选取

对于词向量模型，该特征的特点是稠密、维度小，因此选择 `SVC` 且核函数为 `rbf`。这种方法分类的准确率较高，但算法复杂度较大，适合维度小、稠密的特征。

对于TF-IDF特征，该特征的特点是稀疏、维度大 (17000+ 维度)，因此选择 `LinearSVC` 分类器。因为此时维度太大，`rbf` 的核函数此时已经无法计算，且稀疏的特征一般而言可以很好地被线性分类。

2.3.1.2. D-Tree

决策树的训练直接调用 `sklearn.tree` 库即可。

需要注意的是，决策树在维度大的特征中的训练十分缓慢，因此ID-IDF特征的训练需要较久的时间。

2.3.2. 集成学习算法

2.3.2.1. Bagging算法

随机采样(bootstrap)就是从我们的训练集里面采集固定个数的样本，但是每采集一个样本后，都将样本放回。也就是说，之前采集到的样本在放回后有可能继续被采集到。对于我们的Bagging算法，一般会随机采集和训练集样本数m一样个数的样本。这样得到的采样集和训练集样本的个数相同，但是样本内容不同。如果我们对有m个样本训练集做T次的随机采样，则由于随机性，T个采样集各不相同。

这种算法可以降低方差，但不可以降低bias。

2.3.2.2. AdaBoost.M1

算法流程如下：

Algorithm 4 AdaBoost.M1

Input: Training set $S = \{\mathbf{x}_i, y_i\}, i = 1, \dots, N$; and $y_i \in \mathbb{C}, \mathbb{C} = \{c_1, \dots, c_m\}$; T : Number of iterations; I : Weak learner

Output: Boosted classifier:

$$H(x) = \arg \max_{y \in \mathbb{C}} \sum_{t=1}^T \ln \left(\frac{1}{\beta_t} \right) [h_t(x) = y] \text{ where } h_t, \beta_t$$

are the induced classifiers (with $h_t(x) \in \mathbb{C}$) and their assigned weights, respectively

- 1: $D_1(i) \leftarrow 1/N$ for $i = 1, \dots, N$
- 2: **for** $t = 1$ to T **do**
- 3: $h_t \leftarrow I(S, D_t)$
- 4: $\varepsilon_t \leftarrow \sum_{i=1}^N D_t(i) [h_t(\mathbf{x}_i) \neq y_i]$
- 5: **if** $\varepsilon_t > 0.5$ **then**
- 6: $T \leftarrow t - 1$
- 7: **return**
- 8: **end if**
- 9: $\beta_t = \frac{\varepsilon_t}{1 - \varepsilon_t}$
- 10: $D_{t+1}(i) = D_t(i) \cdot \beta_t^{1 - [h_t(\mathbf{x}_i) \neq y_i]}$ for $i = 1, \dots, N$
- 11: Normalize D_{t+1} to be a proper distribution
- 12: **end for**

<http://blog.csdn.net/tyh70537>

2.4. 预测

2.4.1. Bagging算法

对N个投票器取众数即可。

2.4.2. AdaBoost.M1算法

对N个投票器根据训练时的权重求出带权投票的最大值对应的标签。

三、实验结果

最终最好的结果为TF-IDF + D-Tree + AdaBoost.M1，RMSE为0.35483。我的ID为kissintherain，ranking为53。

实验数据

训练方法 \ RMSE	词向量模型	TF-IDF方法
SVC + Bagging	0.52343	0.45084
SVC + AdaBoost.M1	0.49243	0.43464
SVC (single)	0.52035	0.43549
D-Tree + Bagging	0.38393	0.36549
D-Tree + AdaBoost.M1	0.35494	0.35483
D-Tree (single)	0.48913	0.59423

实验结果分析

1. 对于SVM，包括两种不同的核函数在内，可以看出加入集成学习方法后结果都不如D-Tree。这是因为D-Tree模型稳定性差，在加入集成学习后很明显有很强的增益效果，但SVM较稳定，在集成学习中的增益不明显。
2. 词向量模型都劣于TF-IDF方法。一方面，词模型不够精确，自己训练的由于语料库数据较少，对词的表示不够精确；而网上现有模型都没有口语化的词语，也有一定的偏差。（这一点在实验设计中已经提到。）另一方面，由于文本全为短评，且去除了停等词，TF-IDF在这种较短的言简意赅的句子中有很大的优势。
3. Bagging不如AdaBoost.M1，因为后者更能减少模型的bias，而前者只是降低方差。
4. 最优的组合为TF-IDF + D-Tree + AdaBoost.M1，原因即为以上三点。