数据挖掘大作业实验报告

黄予　　2013011363　　计 34

1　数据预处理

1.1 数据解压

1. 将 nyt_corpus_628804095.zip 解压至 nyt_corpus 文件夹
2. 将 decompress.sh 脚本复制到 nyt_corpus 文件夹下
3. 运行 decompress.sh 脚本即可解压所有文件

1.2 提取文本特征

1. 使用 python 包 xml.etree.cElementTree 读取 xml 文件
2. 以<block class="full_text">为标志获取文本，将除了字母数字之外的字符转换为空格，再将数字替换为#
3. 以 classifier 为标志获取分类标签，一篇文本一般有多个标签

1.3 分类筛选

直接运行 countClassifier.py 可以得到指定年份的分类统计，年份在脚本中指定，共分析了 1987 年、1990 年、1995 年、2000 年、2005 年的分类统计，详见 1987.txt、1990.txt、1995.txt、2000.txt、2005.txt。以 1987 年的部分结果为例，格式为“标签：出现该标签的文本数目”：

```
Top/News :  35533
Top/News/Business :  33625
Top/Features/Travel/Guides/Destinations/North America :  18969
Top/Features/Travel/Guides/Destinations/North America/United States :  18487
Top/News/New York and Region :  13364
Top/Features/Arts :  10870
Top/News/U.S. :  10464
Top/News/Sports :  9575
Top/News/World :  8143
Top/Opinion :  7728
Top/Opinion/Opinion :  7728
Top/Classifieds/Job Market/Job Categories/Banking, Finance and Insurance :
6562
Top/News/U.S./Mid-Atlantic :  6144
Top/Features/Travel/Guides/Destinations/North America/United States/New York/
New York City :  6008
Top/Features/Travel/Guides/Destinations/North America/United States/New York
:  5692
Top/News/U.S./U.S. States, Territories and Possessions/New York :  5415
Top/Features/Style :  4552
Top/Features/Travel/Guides/Destinations/Europe :  4094
Top/Features/Style/Fashion and Style :  3743
Top/Features/Travel/Guides/Destinations/Asia :  3440
Top/Features/Travel/Guides/Destinations/Middle East :  3394
Top/News/Washington :  3334
Top/Features/Travel/Guides/Activities and Interests/Music :  3204
Top/Features/Arts/Music :  3171
Top/News/World/Middle East :  3001
Top/Classifieds/Job Market/Job Categories/Government, Philanthropy and NGO :
2852
Top/Features/Style/Fashion and Style/Weddings and Celebrations :  2711
```

从结果中筛选出各个年份均出现较多的、语意不重合的标签如下：

0. Top/News/Sports
1. Top/News/Business
2. Top/Features/Arts
3. Top/Features/Travel
4. Top/News/World
5. Top/Features/Style
6. Top/News/Health
7. Top/News/Obituaries

与推荐的类别有所不同，去掉了 science、books、home and garden 这三类，因为这三类文本的数目不足。此外，又新加入 Top/News/Health 类，合计 8 个类别。

1.4 文本筛选

在给定的文本集合中：

1. 去掉非上述 8 个分类的文本
2. 去除同时属于多个分类的文本，例如同时含有 Top/News/Sports 标签与 Top/Features/Travel 标签的文本，这样可以避免“交叉文本的干扰”，经测试，可以使准确率等各项评价指标上升 5 个百分点左右。
3. 1987 年的所有文本，经过该处理后，读取的 8 个类的文本数如下：

```
target:  0 , name:  Top/News/Sports , count:  8096
target:  1 , name:  Top/News/Business , count:  28152
target:  2 , name:  Top/Features/Arts , count:  6640
target:  3 , name:  Top/Features/Travel , count:  17750
target:  4 , name:  Top/News/World , count:  1326
target:  5 , name:  Top/Features/Style , count:  3731
target:  6 , name:  Top/News/Health , count:  1453
target:  7 , name:  Top/News/Obituaries , count:  1886
```

4. 1987 年上半年的所有文本，经过该处理后，读取的 8 个类的文本数如下：

```
target:  0 , name:  Top/News/Sports , count:  4091
target:  1 , name:  Top/News/Business , count:  14427
target:  2 , name:  Top/Features/Arts , count:  3407
target:  3 , name:  Top/Features/Travel , count:  8890
target:  4 , name:  Top/News/World , count:  628
target:  5 , name:  Top/Features/Style , count:  1791
target:  6 , name:  Top/News/Health , count:  701
target:  7 , name:  Top/News/Obituaries , count:  827
```

5. 1987 年 1 月的所有文本，经过该处理后，读取的 8 个类的文本数如下：

```
target:  0 , name:  Top/News/Sports , count:  771
target:  1 , name:  Top/News/Business , count:  2475
target:  2 , name:  Top/Features/Arts , count:  533
target:  3 , name:  Top/Features/Travel , count:  1548
target:  4 , name:  Top/News/World , count:  87
target:  5 , name:  Top/Features/Style , count:  251
target:  6 , name:  Top/News/Health , count:  111
target:  7 , name:  Top/News/Obituaries , count:  155
```

1.5 去除停用词

停用词是指一些没有实际含义的功能词，在各个文本中均大量出现，对分类没有实际帮助的一类词语。程序可以使用以下两个停用词表：

1. sklearn.feature_extraction.text.CountVectorizer 的默认停用词表：

```
['all', 'six', 'less', 'being', 'indeed', 'over', 'move', 'anyway', 'four', 'not', 'own', 'through',
'yourselves', 'fify', 'where', 'mill', 'only', 'find', 'before', 'one', 'whose', 'system', 'how',
'somewhere', 'with', 'thick', 'show', 'had', 'enough', 'should', 'to', 'must', 'whom', 'seeming', 'under',
'ours', 'has', 'might', 'thereafter', 'latterly', 'do', 'them', 'his', 'around', 'than', 'get', 'very', 'de',
'none', 'cannot', 'every', 'whether', 'they', 'front', 'during', 'thus', 'now', 'him', 'nor', 'name',
'several', 'hereafter', 'always', 'who', 'cry', 'whither', 'this', 'someone', 'either', 'each', 'become',
'thereupon', 'sometime', 'side', 'two', 'therein', 'twelve', 'because', 'often', 'ten', 'our', 'eg', 'some',
'back', 'up', 'go', 'namely', 'towards', 'are', 'further', 'beyond', 'ourselves', 'yet', 'out', 'even',
'will', 'what', 'still', 'for', 'bottom', 'mine', 'since', 'please', 'forty', 'per', 'its', 'everything',
'behind', 'un', 'above', 'between', 'it', 'neither', 'seemed', 'ever', 'across', 'she', 'somehow', 'be',
'we', 'full', 'never', 'sixty', 'however', 'here', 'otherwise', 'were', 'whereupon', 'nowhere', 'although',
'found', 'alone', 're', 'along', 'fifteen', 'by', 'both', 'about', 'last', 'would', 'anything', 'via',
'many', 'could', 'thence', 'put', 'against', 'keep', 'etc', 'amount', 'became', 'ltd', 'hence', 'onto', 'or',
'con', 'among', 'already', 'co', 'afterwards', 'formerly', 'within', 'seems', 'into', 'others', 'while',
'whatever', 'except', 'down', 'hers', 'everyone', 'done', 'least', 'another', 'whoever', 'moreover',
'couldnt', 'throughout', 'anyhow', 'yourself', 'three', 'from', 'her', 'few', 'together', 'top', 'there',
'due', 'been', 'next', 'anyone', 'eleven', 'much', 'call', 'therefore', 'interest', 'then', 'thru',
'themselves', 'hundred', 'was', 'sincere', 'empty', 'more', 'himself', 'elsewhere', 'mostly', 'on', 'fire',
'am', 'becoming', 'hereby', 'amongst', 'else', 'part', 'everywhere', 'too', 'herself', 'former', 'those',
'he', 'me', 'myself', 'made', 'twenty', 'these', 'bill', 'cant', 'us', 'until', 'besides', 'nevertheless',
'below', 'anywhere', 'nine', 'can', 'of', 'your', 'toward', 'my', 'something', 'and', 'whereafter',
'whenever', 'give', 'almost', 'wherever', 'is', 'describe', 'beforehand', 'herein', 'an', 'as', 'itself',
'at', 'have', 'in', 'seem', 'whence', 'ie', 'any', 'fill', 'again', 'hasnt', 'inc', 'thereby', 'thin', 'no',
'perhaps', 'latter', 'meanwhile', 'when', 'detail', 'same', 'wherein', 'beside', 'also', 'that', 'other',
'take', 'which', 'becomes', 'you', 'if', 'nobody', 'see', 'though', 'may', 'after', 'upon', 'most',
'hereupon', 'eight', 'but', 'serious', 'nothing', 'such', 'why', 'a', 'off', 'whereby', 'third', 'i',
'whole', 'noone', 'sometimes', 'well', 'amoungst', 'yours', 'their', 'rather', 'without', 'so', 'five',
'the', 'first', 'whereas', 'once']
```

2. http://www.ranks.nl/stopwords 下载的 Long Stop List，放置于 longStopWords.json 文件中。

由于 Long Stop Words 的单词数目较多，使得生成 tf-idf 矩阵的速度显著降低（大约降低 10 倍），而各项评价指标的上升程度并不高，故程序使用默认停用词表。可以在程序中通过注释相应代码更改配置，具体见 classify.py 与 cluster.py：

```
#stopWords = json.load(file('longStopWords.json'))      #提取tf-idf矩阵较慢，是使用默认词表速度的10倍
stopWords = 'english'
```

1.6 获得 tf-idf 矩阵

将所有获得的文本转换为 tf-idf 矩阵，行为文本，列为单词，即每个文本对应一个 tf-idf 单词向量，用该矩阵作为输入，训练模型。以 1987 年为例，经过 1.4 以及 1.5 的处理，提取的矩阵大小为：

```
tf-idf matrix shape = (文本数, 单词数) =  (69034, 173779)
```

共读取 69034 个文本，单词数目为 173779。因为单词数目庞大，故在分类时在本机上至多只能使用 1 年的数据进行训练，否则进程会由于内存不够而终止。

2 分类

2.1 说明

1. 为节省时间，程序采用 5 折分层交叉验证，而不是 10 折分层交叉验证。
2. 分类使用准确率（accuracy）、精度（precision）、召回率（recall）、f1 值（f1）、ROC 曲线、AUC 值（area under roc curve）作为分类的评价指标，最后的结果均使用交叉验证的平均值。
3. ROC 曲线使用交叉验证的第一折的结果作图，分别以 8 个类作为正例，作 8 条曲线。
4. 以下分类结果的训练数据均采用 1987 年的文本
5. 可以在 classify_settings.json 文件中设置读取哪一文件夹下的文本，以及设置分类。
6. 由于 SVM 耗时太长，故放弃。
7. 程序清单：classify.py，mylib.py，countClassifier.py

2.2 Logistic Regression

1. 使用模型、参数设置以及耗时

   (1) 模型：sklearn.linear_model.LogisticRegression

   (2) 参数设置：C = 40.0, penalty = 'l2', solver = 'newton-cg',其余为默认参数

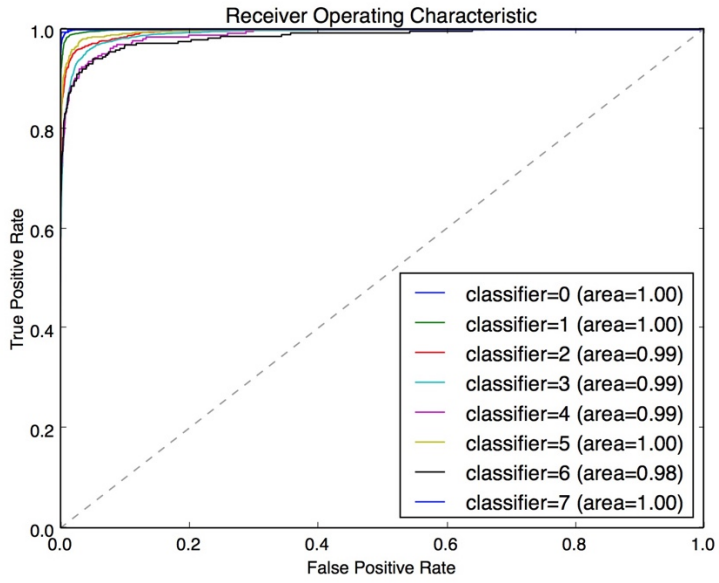
   (3) 运行耗时：0:07:20.391508

2. 最终结果

   (1) 各项评价指标

accuracy	precision	recall	f1	AUC
0.947953	0.948140	0.947953	0.947471	0.994797

   (2) ROC 曲线

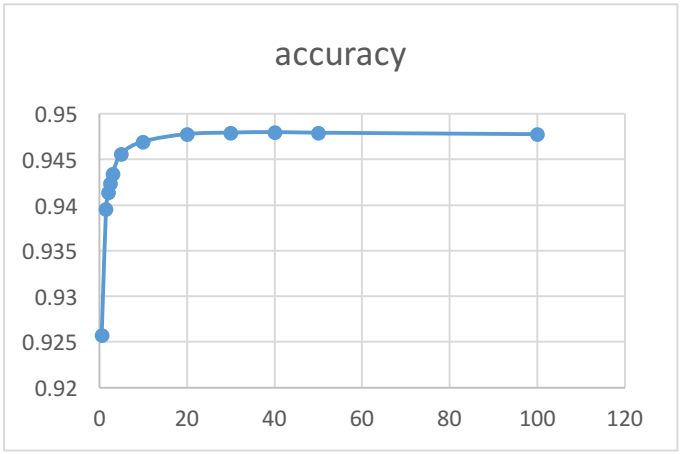


   (3) 结果分析

   逻辑回归是所采用的分类模型中最好的，准确率能达到 94.8%，ROC 曲线的“上凸”程度达到最大，结果非常让人满意。

3. C 值对准确率的影响

C	accuracy
0.5	0.925732
1.5	0.939580
2.0	0.941391
2.5	0.942376
3.0	0.943434
5.0	0.945621
10.0	0.946954
20.0	0.947765
30.0	0.947924
40.0	0.947982
50.0	0.947910
100.0	0.947750



从散点图可以看出，准确率随 C 值的增大而增大，在达到 20 之前增大速度较快，在超过 20 之后趋于平缓，程序最后采用的 C 值为 40。

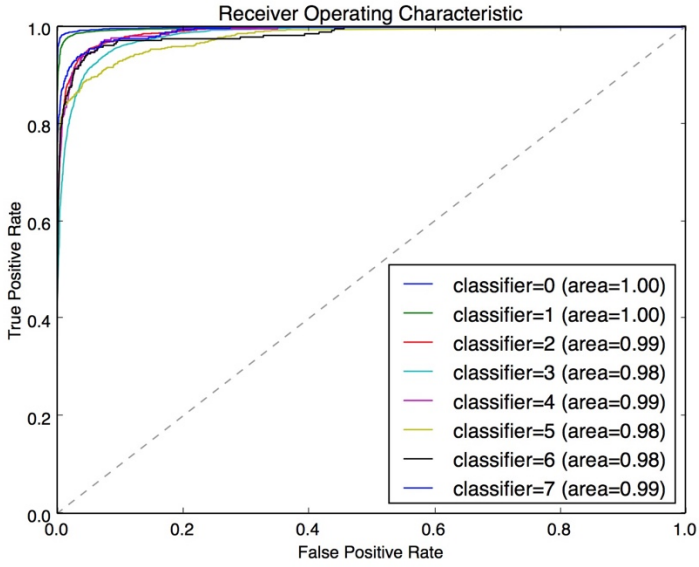
2.3 Naïve Bayes

1. 使用模型、参数设置以及耗时
   (1) 模型：sklearn.naive_bayes.MultinomialNB
   (2) 参数设置：alpha = 0.01，其余为默认值
   (3) 运行耗时：0:00:03.754291

2. 最终结果
   (1) 各项评价指标：

accuracy	precision	recall	f1	AUC
0.914462	0.919335	0.914462	0.915457	0.988802

   (2) ROC 曲线：



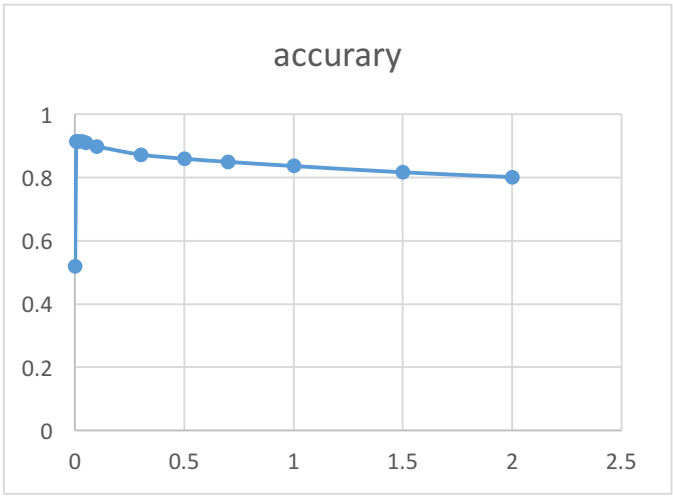
(3) 结果分析

各项评价标准均达到 91%以上，ROC 曲线也非常美观，可以说结果不错。

3. alpha 值对准确率的影响

在调参的过程中，发现 alpha 值对结果的影响较大，下面以其对准确率的影响进行分析，对其他指标的影响相似：

alpha	accuracy
0	0.519
0.005	0.913
0.01	0.914
0.03	0.913
0.05	0.909
0.1	0.898
0.3	0.872
0.5	0.859
0.7	0.849
1	0.836
1.5	0.816
2	0.801



在 alpha 为 0.01 附近时，分类准确率达到最大值，越远离 0.01 附近，则准确率递减。程序最后采用的 alpha 值为 0.01。

2.4 Random Forest

1. 使用模型、参数设置以及耗时

(1) 模型：sklearn.ensemble.RandomForestClassifier

(2) 参数设置：n_estimators = 60, n_jobs = -1，其余为默认参数。n_estimators 指随机森林中树的个数，n_jobs = -1 指使用所有的 CPU 训练模型。

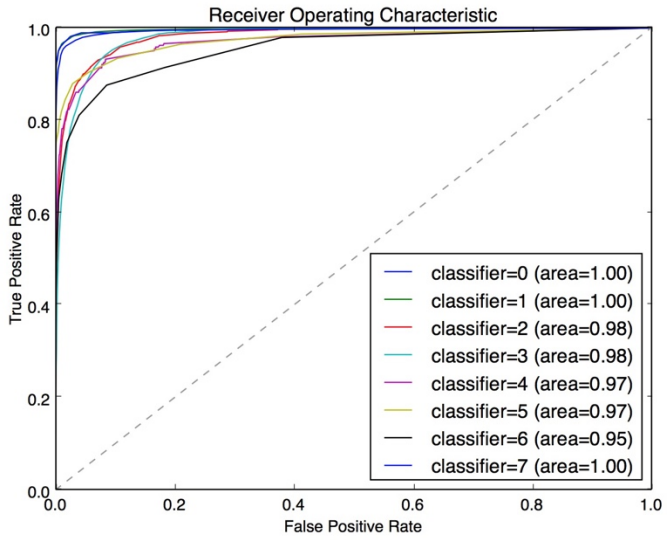
(3) 运行耗时：0:04:32.550672

2. 最终结果

(1) 各项评价指标：

accuracy	precision	recall	f1	AUC
0.887331	0.907774	0.887331	0.878312	0.983333

(2) ROC 曲线

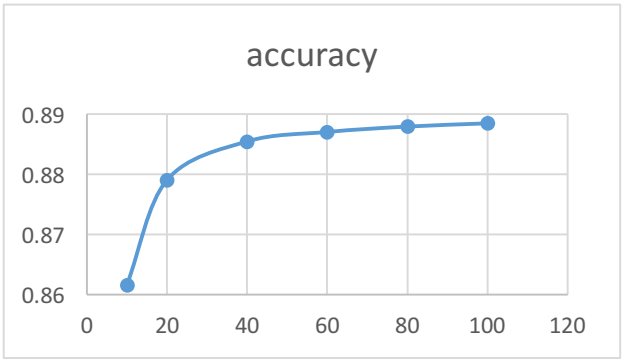


(3) 结果分析

准确率等各项评价指标（除去 AUC）在 88%左右，与前两者相比差了一些。

3. n_estimators 对准确率的影响

n_estimators	accuracy
10	0.861648
20	0.879060
40	0.885492
60	0.887070
80	0.887969
100	0.888504



从散点图可以看出，准确率随 n_estimators 值的增大而增大，在 40 之前速度较快，40 之后趋于平缓，程序最后采用的 n_estimators 值为 40。

2.5 Decision Tree

1. 使用模型、参数设置以及耗时

(1) 模型：sklearn.tree. DecisionTreeClassifier

(2) 参数设置：使用默认参数

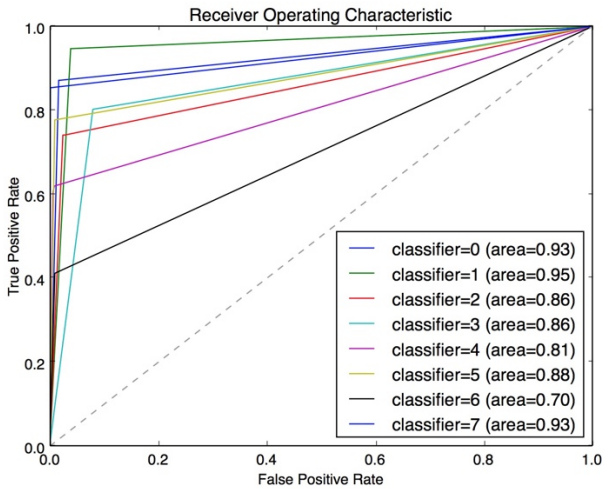
(3) 运行耗时：1:11:15.799324

2. 最终结果

(1) 各项评价指标

accuracy	precision	recall	f1	AUC
0.854724	0.854048	0.854724	0.854233	0.870032

(2) ROC 曲线



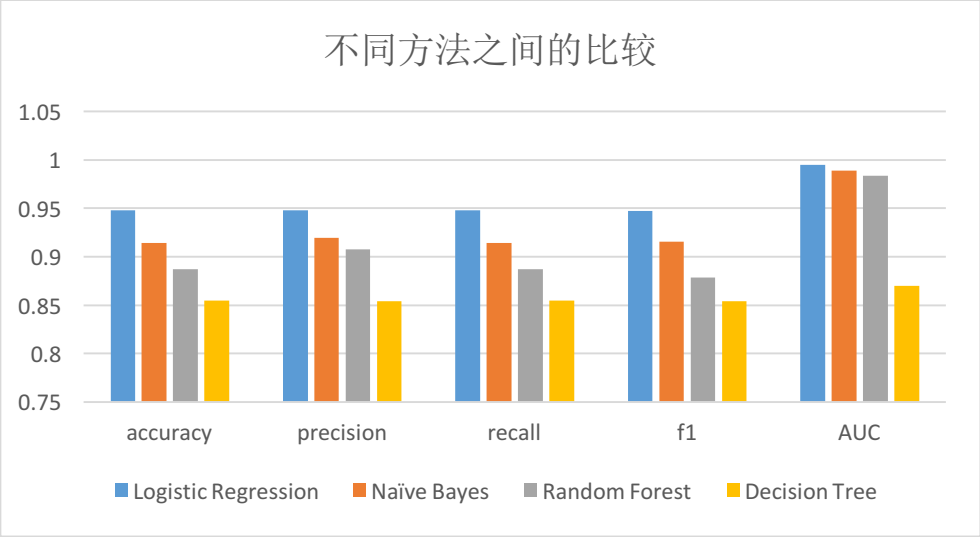
(3) 结果分析

准确率等各项评价指标（除去 AUC）在 85%左右，ROC 曲线较差。

2.6 不同方法之间的比较

1. 各项评价指标

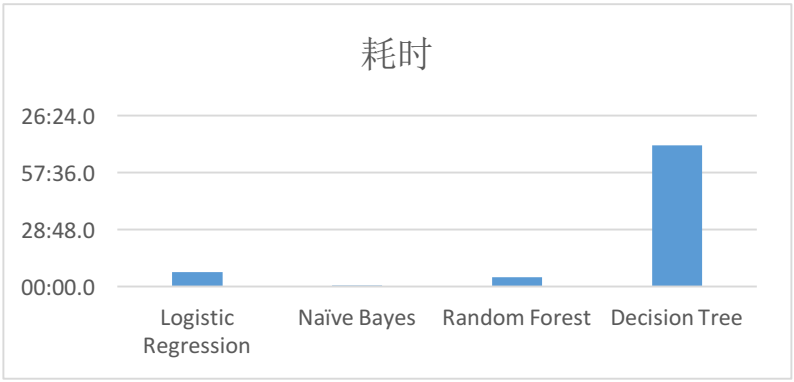
	accuracy	precision	recall	f1	AUC
Logistic Regression	0.947953	0.948140	0.947953	0.947471	0.994797
Naïve Bayes	0.914462	0.919335	0.914462	0.915457	0.988802
Random Forest	0.887331	0.907774	0.887331	0.878312	0.983333
Decision Tree	0.854724	0.854048	0.854724	0.854233	0.870032



就方法的效果而言，从好到坏依次为 Logistic Regression、Naïve Bayes、Random Forest、Decision Tree，最好结果与最差结果大约相差 10 个百分点。

2. 运行耗时

方法	Logistic Regression	Naïve Bayes	Random Forest	Decision Tree
耗时	0:07:20.391508	0:00:03.754291	0:04:32.550672	1:11:15.799324



就方法的耗时而言，从好到坏依次是 Naïve Bayes、Random Forest、Logistic Regression、Decision Tree，其中 Decision Tree 的耗时远超其他三者。值得一提的是，Naïve Bayes 原理简单，速度非常快，并且效果还不错，性价比很高。

3 聚类

3.1 说明

1. 聚类的评价指标采用 Adjusted Mutual Information(简称 AMI)、Adjusted Rand Index(简称 ARI)、Homogeneity、Completeness 以及 V-Measure。
2. 多维数据可视化采用了 PCA 降维的方法，将多维数据映射到 2 维空间。由于进行 PCA 降维需要将稀疏矩阵转换为稠密矩阵，且 PCA 降维比较耗时，故进行作图时使用 1 个月的数据。
3. 依旧采用数据预处理中所得的 8 类文本作为聚类的训练数据，聚类算法的聚类数目也都设置为 8。

4. 可以在 cluster_settings.json 文件中设置读取哪一文件夹下的文本，以及读取的分类。
5. 程序清单：cluster.py， mylib.py

3.2 K-Means

1. 模型以及参数设置
   (1) sklearn.decomposition.IncrementalPCA
   (2) 参数设置：n_clusters = 8, max_iter = 600, tol = 1e-5, n_jobs = -1， max_iter 是指最大迭代次数，tol 为判断是否收敛的一个阈值，n_jobs 指使用所有 CPU。

2. 使用 1987 年的文本训练

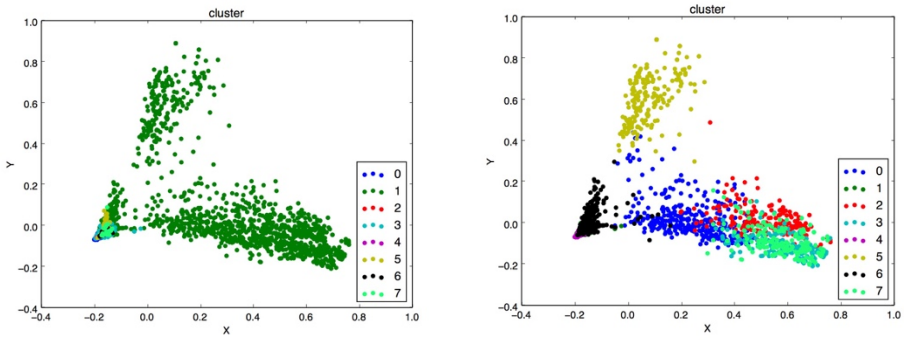
AMI	ARI	Homogeneity	Completeness	V-Measure
0.430926	0.145408	0.431049	0.454591	0.442507

除去 ARI 较低以外，各项指标大多在 0.44 左右，与分类相比，聚类的效果并不是很理想。

3. 使用 1987 年 1 月的文本训练

AMI	ARI	Homogeneity	Completeness	V-Measure
0.343123	0.049079	0.359989	0.344812	0.352237

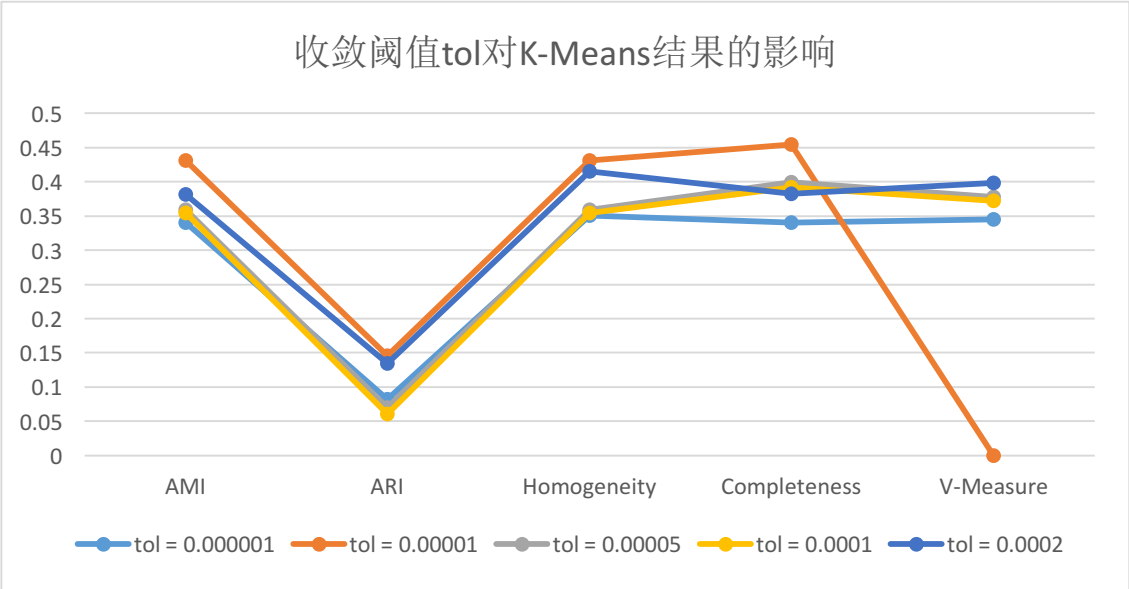
由于训练数据少，与使用 1987 年整年的数据进行训练相比，各项指标均下降 0.1 左右，进行 PCA 降维后数据可视化如下：



左图为真实的分类，右图为 K-Means 算法给出的聚类。从左图可以看出同一类文本的 tf-idf 向量的聚合程度不一定高，不同类文本的向量重合程度非常高，但聚类效果不理想的原因可以窥得一二。右图为 K-Means 算法给出的划分，个人认为还是十分合理的，虽然与真实数据差别较大。

4. 收敛阈值 tol 对 K-Means 结果的影响

tol	AMI	ARI	Homogeneity	Completeness	V-Measure
1e-6	0.339894	0.081827	0.350649	0.340032	0.345259
1e-5	0.430907	0.145385	0.431030	0.454579	0. 442491
5e-5	0.358736	0.069991	0.358874	0.398887	0.377824
1e-4	0.354316	0.060878	0.354455	0.392107	0.372332
2e-4	0.381965	0.134226	0.415458	0.382088	0.398075



训练数据为 1987 年整年数据，可见 tol 值为 1e-5 时最优，故程序采用的 tol 值为 1e-5。

3.3 DBSCAN

1. 模型以及参数设置
   (1) 模型：sklearn.cluster.DBSCAN
   (2) 参数配置: eps = 0.99, min_samples=5，eps 指邻域的半径，min_samples 指判断一个点是否为核心对象时使用的邻域内最少点数。

2. 使用 1987 年上半年的文本训练

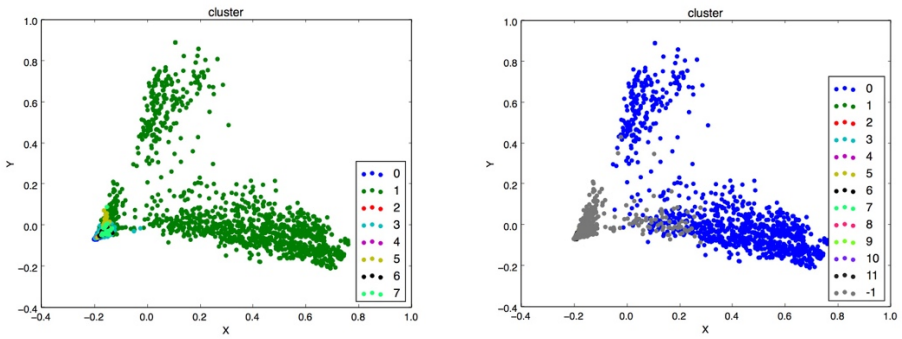
AMI	ARI	Homogeneity	Completeness	V-Measure
0.286637	0.114999	0.297621	0.333895	0.314717

由于 scipy 在做稀疏矩阵乘法时要求稀疏矩阵的非零元不能过多，因此只能使用 1987 年上半年的文本进行训练，而无法使用一整年。从结果上看，除去 ARI 较低之外，各项指标大多在 0.3 左右，结果并不是很理想。

3. 使用 1987 年 1 月的文本训练

AMI	ARI	Homogeneity	Completeness	V-Measure
0.238223	0.101548	0.246321	0.411254	0.308103

由于训练数据少，与使用 1987 年上半年的数据进行训练相比，除去 Completeness 之外，各项指标均有所下降，进行 PCA 降维后数据可视化如下：



左图为真实的分类，右图为 DBSCAN 算法给出的分类，由于 sklearn 的 DBSCAN 模型并未给出最大聚类数目的接口，故聚类结果可能不等于 8 类。从右图可以看出，类 1（Top/News/Business）被聚类得较好，而其他类则大多被分为了噪点，效果不是很理想。

3.4 Agglomerative Clustering

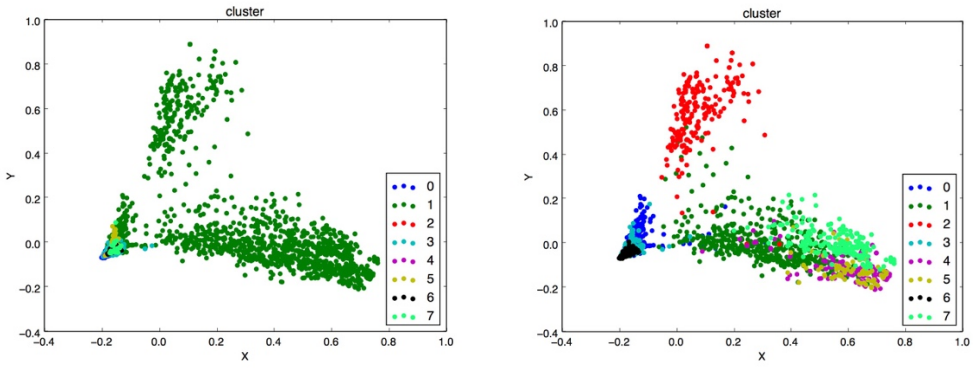
1. 模型以及参数设置
   (1) 模型：sklearn.cluster.AgglomerativeClustering
   (2) 参数设置：使用默认参数

2. 使用 1987 年 1 月的文本训练

AMI	ARI	Homogeneity	Completeness	V-Measure
0.405043	0.143588	0.413888	0.406616	0.410220

由于 AgglomerativeClustering 提供的接口要求必须使用稠密矩阵，因此需要将 tf-idf 稀疏矩阵类型转换为稠密矩阵类型进行计算，若是使用一年的数据量则内存不足，所以这里只使用 1 个月的数据进行训练

除去 ARI，各项评价指标的得分在 0.4 左右，结果较前两种聚类方法都要好。进行 PCA 降维后可视化如下：



聚类的结果（右图）与 K-Means 的结果很相像，但是左下角那一块明显要比 K-Means 要好，K-Means 中左下角这一块几乎都被分到同一类中，而 AgglomerativeClustering 则将左下角区分开来，并且与原始数据分类相比很相像，因此各项指标均比 K-Means 要好。