Task01

1. 赛题介绍

- 数据类型：新闻类文本数据（已经脱敏处理）
- 分类类别：{'科技': 0, '股票': 1, '体育': 2, '娱乐': 3, '时政': 4, '社会': 5, '教育': 6, '财经': 7, '家居': 8, '游戏': 9, '房产': 10, '时尚': 11, '彩票': 12, '星座': 13}
- 数据：
  - 训练集：train_set.csv　　200000 条记录
  - 测试集：test_a.csv　　50000 条记录
- 评判标准：
  - F1-score
  - $$F1 = 2 * \frac{precision * recall}{precision + recall}$$

2.解题方案

- 思路：特征提取（词向量） + 分类模型
  - 特征提取：word2vec、TF-IDF、FastTest
  - 分类模型: 机器学习模型、深度学习模型

2.1. 思路1：TF-IDF + 机器学习分类器

直接使用TF-IDF对文本提取特征，并使用分类器进行分类。在分类器的选择上，可以使用SVM、LR、或者XGBoost。

- **TF-IDF** 是什么？能解决什么问题？
  - TF-IDF (term frequency-inverse document frequency) 技术出现的原因是因为以 **计数特征**（one-hot, one-hot based count）　文本向量化的方法的不足而产生的。
  - 在以计数特征文本分词并向量化后，我们可以得到词汇表中每个词在各个文本中形成的词向量。
  - 举个语料库 corpus=["我 来到 风景 非常 美丽 的 杭州 喝 到了 非常 好喝 的 龙井",

    "我 非常 喜欢 旅游",

    "我 非常 喜欢 吃 苹果 ",

    "我 非常 喜欢 看 电视"]
  - 统计上述语料库的词频，会发现 “非常这个词” 从计数特征来说，似乎与文本的更加具有联系（出现次数多），但实际 “ 非常”这个词十分普遍，在四个文本中都出现了，重要性很却没有词频较低的 “杭州” 重要，我们的 IDF 就是来帮助我们来反应这个词的重要性的，进而修正仅仅用词频表示的词特征值。

- 概括来讲，IDF反应了一个词在所有文本中出现的频率，如果一个词在很多的文本中出现，那么它的IDF值应该低，比如上文中的“非常”。而反过来如果一个词在比较少的文本中出现，那么它的IDF值应该高。一个极端的情况，如果一个词在所有的文本中都出现，那么它的IDF值应该为0。

  公式：

  词频（Term Frequency, TF）表示关键词 w 在文档 D_i 中出现的频率：

  $$TF_{w,D_i} = \frac{count(w)}{|D_i|}$$

  其中，$count(w)$ 为关键词w的出现次数，$|D_i|$为文档 D_i 中所有词的数量。

  逆文档频率（Inverse Document Frequency, IDF）反映 关键词的普遍程度——当一个词越普遍（即有大量文档包含这个词）时，其IDF值越低；反之，则IDF值越高。IDF定义如下：

  $$IDF_w = log(\frac{N}{\sum_{i=1}^{N} I(w, Di)})$$

  其中，N 为所有文档总数， $I(w, D_i)$ 表示文档 D_i 是否包含关键词，包含为1， 不包含为0.

  当然为了避免 IDF 中公式中的分母为0，因此做平滑处理：

  $$IDF_w = log(\frac{N}{1 + \sum_{i=1}^{N} I(w, Di)})$$

  即，关键词 w 在文档 D_i 的 $TF - IDF$ 值为：

  $$TF - IDF_{w,D_i} = TF_{w,D_i} * IDF_w$$

  【结论】：当一个词在文档**频率越高**并且**新鲜度高**（即普遍度低），其TF-IDF值越高。

  TF-IDF兼顾词频与新鲜度，过滤一些常见词，保留能提供更多信息的重要词。

- TF-IDF + LR 实战操作：

新闻文本分类实战

```
import os
print(os.getcwd())
```

```
C:\Users\13398\code\datawhale\NLP
```

1.导入需要的包

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
```

```
train = pd.read_csv("./input/train_set.csv",sep = '\t')
```

```
test = pd.read_csv("./input/test_a.csv", sep = '\t')
```

```
train_text = train['text']
test_text = test['text']

test_text
```

```
0        5399 3117 1070 4321 4568 2621 5466 3772 4516 2...
1        2491 4109 1757 7539 648 3695 3038 4490 23 7019...
2        2673 5076 6835 2835 5948 5677 3247 4124 2465 5...
3        4562 4893 2210 4761 3659 1324 2595 5949 4583 2...
4        4269 7134 2614 1724 4464 1324 3370 3370 2106 2...
                              ...
49995    3725 4498 2282 1647 6293 4245 4498 3615 1141 2...
49996    4811 465 3800 1394 3038 2376 2327 5165 3070 57...
49997    5338 1952 3117 4109 299 6656 6654 3792 6831 21...
49998    893 3469 5775 584 2490 4223 6569 6663 2124 168...
49999    2400 4409 4412 2210 5122 4464 7186 2465 1327 9...
Name: text, Length: 50000, dtype: object
```

将训练集和测试集合并处理，避免不统一

```
all_text = pd.concat([train_text, test_text])
```

```
all_text
```

```
0        2967 6758 339 2021 1854 3731 4109 3792 4149 15...
1        4464 486 6352 5619 2465 4802 1452 3137 5778 54...
2        7346 4068 5074 3747 5681 6093 1777 2226 7354 6...
3        7159 948 4866 2109 5520 2490 211 3956 5520 549...
4        3646 3055 3055 2490 4659 6065 3370 5814 2465 5...
                              ...
49995    3725 4498 2282 1647 6293 4245 4498 3615 1141 2...
49996    4811 465 3800 1394 3038 2376 2327 5165 3070 57...
49997    5338 1952 3117 4109 299 6656 6654 3792 6831 21...
49998    893 3469 5775 584 2490 4223 6569 6663 2124 168...
49999    2400 4409 4412 2210 5122 4464 7186 2465 1327 9...
Name: text, Length: 250000, dtype: object
```

```
tf_idf = TfidfVectorizer()
tf_idf.fit(all_text)
train_w2v = tf_idf.transform(train_text)
test_w2v = tf_idf.transform(test_text)
train_w2v
```

```
<200000x6967 sparse matrix of type '<class 'numpy.float64'>'
    with 56068855 stored elements in Compressed Sparse Row format>
```

2.logistic regression

```
X_train = train_w2v
y_train = train['label']

# 切分验证集
x_train_set, x_valid_set, y_train_set, y_valid_set =
train_test_split(X_train, y_train, test_size = 0.3)
x_test_set = test_w2v
```

```
clf = LogisticRegression(C=4, n_jobs=16)
clf.fit(x_train_set, y_train_set)

y_pred = clf.predict(x_valid_set)
train_scores = clf.score(x_train_set, y_train_set)
print(train_scores, f1_score(y_pred, y_valid_set, average = 'macro'))
```

```
0.9403928571428571 0.9110279413956893
```

```
y_pred_final = clf.predict(x_test_set)

submission = pd.read_csv('./input/test_a_sample_submit.csv')
submission['label'] = y_pred_final
submission.to_csv('./lr_submission.csv', index=False)
```

成绩