NLP新闻分类学习赛

> https://tianchi.aliyun.com/competition/entrance/531810/forum

介绍

这只是一个经典的分类任务，从不等长的文本序列，得出一个新闻的分类，一共是13种新闻分类。

从分类算法的发展历史来说，这有800种方法可以完成

- 传统的机器学习算法
- CNN网络
- 新的预训练模型，基于Transformer结构的网络

数据

label	text
6	57 44 66 56 2 3 3 37 5 41 9 57 44 47 45 33 13 63 58 31 17 47 0 1 1 69 26 60 62 15 21 12 49 18 38 20 50 23 57 44 45 33 25 28 47 22 52 35 30 14 24 69 54 7 48 19 11 51 16 43 26 34 53 27 64 8 4 42 36 46 65 69 29 39 15 37 57 44 45 33 69 54 7 25 40 35 30 66 56 47 55 69 61 10 60 42 36 46 65 37 5 41 32 67 6 59 47 0 1 1 68

训练数据是长这样的。就是把汉字和标点替换成数字的匿名化处理。label是分类。这种匿名化实际上是没有什么用的，可以通过字频分析给它还原到原本的汉字。密码学真的天下第一。

这是标签

> {'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}

NLP分类的基本步骤

做CV那么久，NLP数据的预处理真的是非常麻烦。

向量化

- 可以使用One-hot编码
- 也可以用什么all_data2fold做等分
- 反正最后要放等长的Tensors进入网络

这代码也是叽里咕噜不知道在做什么东西。。。

分等长的10份，再做了一个index和text的字典。。。

拆分训练、验证集

从上述等分的fold中取几个为训练集

词汇表Vocab

词汇表构建（Vocabulary Building）是文本数据预处理中的关键步骤，涉及从训练语料中生成一个包含所有可识别单词、子词或字符的集合。

这是一个预处理的经典步骤，输入是所有句子，输出是一个字典

创建 label 和 index 对应的字典。

具体步骤如下：

- 创建 词 和 `index` 对应的字典，这里包括 2 份字典，分别是：`_id2word` 和 `_id2extword`。
- 其中 `_id2word` 是从新闻得到的， 把词频小于 5 的词替换为了 `UNK`。对应到模型输入的 `batch_inputs1`。
- 也有选择N个最大的词频做词汇表的做法。
- `_id2extword` 是从 `word2vec.txt` 中得到的，有 5976 个词。对应到模型输入的 `batch_inputs2`。
- 后面会有两个 `embedding` 层，其中 `_id2word` 对应的 `embedding` 是可学习的，`_id2extword` 对应的 `embedding` 是从文件中加载的，是固定的。
- 创建 label 和 index 对应的字典。
- 上面这些字典，都是基于`train_data`创建的。
- **常见工具**: 使用工具或库（如 Python 的 `collections.Counter` 或 `NLTK`）来统计词频。过小的词频就去掉。

一个去除了过小词汇的字典。

词汇表构建的关键在于从训练数据中提取并保存最常见的词汇，处理未登录词，并将词汇表映射到索引。在实际应用中，选择合适的词汇表大小和处理策略对于模型的性能至关重要。

网络

别看花里胡哨的写一堆

涉及时间序列的分类就用LSTM，BERT，不涉及的就可以不用这种时序网络。

中间步骤可以很复杂。

如果使用了预训练模型BERT，那么需要在它的基础上做微调，因为数据集做了匿名化处理，数据的分布是不受影响的，但是数据的表征有被影响。

最后保证输入输出的大小正确就大差不差了。

使用了BERT的模型还在跑

这是训练截图

在25W的数据集上跑了一个epoch。在测试集上面遇到了一个困难，主要是忘记加进度条了，失管失控的感觉，跑测试集3个小时出不来?

```
2025-12-09 10:52:38,624 INFO: Total 9000 docs.
2025-12-09 10:52:42,879 INFO: Total 1000 docs.
2025-12-09 10:56:26,257 INFO: Total 50000 docs.
```

```
/opt/conda/lib/python3.7/site-packages/transformers/optimization.py:310:
FutureWarning: This implementation of AdamW is deprecated and will be removed in
a future version. Use the PyTorch implementation torch.optim.AdamW instead, or
set `no_deprecation_warning=True` to disable this warning
  FutureWarning,
2025-12-09 10:56:26,266 INFO: Start training...
/opt/conda/lib/python3.7/site-packages/torch/optim/lr_scheduler.py:249:
UserWarning: To get the last learning rate computed by the scheduler, please use
`get_last_lr()`.
  warnings.warn("To get the last learning rate computed by the scheduler, "
2025-12-09 11:03:24,287 INFO: | epoch   1 | step  50 | batch  50/2250 | lr
0.00020 0.00005 | loss 2.2534 | s/batch 8.36
2025-12-09 11:09:59,876 INFO: | epoch   1 | step 100 | batch 100/2250 | lr
0.00020 0.00005 | loss 1.6283 | s/batch 7.91
2025-12-09 11:16:45,276 INFO: | epoch   1 | step 150 | batch 150/2250 | lr
0.00020 0.00005 | loss 1.3698 | s/batch 8.11
2025-12-09 11:23:36,979 INFO: | epoch   1 | step 200 | batch 200/2250 | lr
0.00020 0.00005 | loss 1.1478 | s/batch 8.23
2025-12-09 11:29:48,576 INFO: | epoch   1 | step 250 | batch 250/2250 | lr
0.00020 0.00004 | loss 0.9757 | s/batch 7.43
2025-12-09 11:37:00,677 INFO: | epoch   1 | step 300 | batch 300/2250 | lr
0.00020 0.00004 | loss 1.0031 | s/batch 8.64
2025-12-09 11:43:46,676 INFO: | epoch   1 | step 350 | batch 350/2250 | lr
0.00020 0.00004 | loss 0.8812 | s/batch 8.12
2025-12-09 11:50:58,375 INFO: | epoch   1 | step 400 | batch 400/2250 | lr
0.00020 0.00004 | loss 0.9257 | s/batch 8.63
2025-12-09 11:57:38,275 INFO: | epoch   1 | step 450 | batch 450/2250 | lr
0.00020 0.00004 | loss 0.8207 | s/batch 8.00
2025-12-09 12:04:37,875 INFO: | epoch   1 | step 500 | batch 500/2250 | lr
0.00020 0.00004 | loss 0.8205 | s/batch 8.39
2025-12-09 12:11:32,378 INFO: | epoch   1 | step 550 | batch 550/2250 | lr
0.00020 0.00004 | loss 0.7561 | s/batch 8.29
2025-12-09 12:18:04,378 INFO: | epoch   1 | step 600 | batch 600/2250 | lr
0.00020 0.00004 | loss 0.7443 | s/batch 7.84
2025-12-09 12:24:34,876 INFO: | epoch   1 | step 650 | batch 650/2250 | lr
0.00020 0.00004 | loss 0.7884 | s/batch 7.81
2025-12-09 12:31:13,885 INFO: | epoch   1 | step 700 | batch 700/2250 | lr
0.00020 0.00003 | loss 0.5986 | s/batch 7.98
2025-12-09 12:38:11,279 INFO: | epoch   1 | step 750 | batch 750/2250 | lr
0.00020 0.00003 | loss 0.5782 | s/batch 8.35
2025-12-09 12:44:46,579 INFO: | epoch   1 | step 800 | batch 800/2250 | lr
0.00020 0.00003 | loss 0.8472 | s/batch 7.91
2025-12-09 12:51:18,275 INFO: | epoch   1 | step 850 | batch 850/2250 | lr
0.00020 0.00003 | loss 0.7664 | s/batch 7.83
2025-12-09 12:58:21,478 INFO: | epoch   1 | step 900 | batch 900/2250 | lr
0.00020 0.00003 | loss 0.4755 | s/batch 8.46
2025-12-09 13:05:04,779 INFO: | epoch   1 | step 950 | batch 950/2250 | lr
0.00020 0.00003 | loss 0.3904 | s/batch 8.07
2025-12-09 13:12:01,576 INFO: | epoch   1 | step 1000 | batch 1000/2250 | lr
0.00015 0.00003 | loss 0.5702 | s/batch 8.34
2025-12-09 13:18:56,877 INFO: | epoch   1 | step 1050 | batch 1050/2250 | lr
0.00015 0.00003 | loss 0.5170 | s/batch 8.31
2025-12-09 13:25:26,779 INFO: | epoch   1 | step 1100 | batch 1100/2250 | lr
0.00015 0.00003 | loss 0.6462 | s/batch 7.80
2025-12-09 13:31:59,577 INFO: | epoch   1 | step 1150 | batch 1150/2250 | lr
0.00015 0.00002 | loss 0.3966 | s/batch 7.85
```

```
2025-12-09 13:39:09,277 INFO: | epoch   1 | step 1200 | batch 1200/2250 | lr
0.00015 0.00002 | loss 0.6802 | s/batch 8.59
2025-12-09 13:45:08,176 INFO: | epoch   1 | step 1250 | batch 1250/2250 | lr
0.00015 0.00002 | loss 0.6326 | s/batch 7.18
2025-12-09 13:51:41,377 INFO: | epoch   1 | step 1300 | batch 1300/2250 | lr
0.00015 0.00002 | loss 0.8273 | s/batch 7.86
2025-12-09 13:58:39,876 INFO: | epoch   1 | step 1350 | batch 1350/2250 | lr
0.00015 0.00002 | loss 0.3733 | s/batch 8.37
2025-12-09 14:05:35,978 INFO: | epoch   1 | step 1400 | batch 1400/2250 | lr
0.00015 0.00002 | loss 0.4661 | s/batch 8.32
2025-12-09 14:12:10,277 INFO: | epoch   1 | step 1450 | batch 1450/2250 | lr
0.00015 0.00002 | loss 0.5738 | s/batch 7.89
2025-12-09 14:19:25,878 INFO: | epoch   1 | step 1500 | batch 1500/2250 | lr
0.00015 0.00002 | loss 0.4779 | s/batch 8.71
2025-12-09 14:25:56,177 INFO: | epoch   1 | step 1550 | batch 1550/2250 | lr
0.00015 0.00002 | loss 0.6068 | s/batch 7.81
2025-12-09 14:32:29,376 INFO: | epoch   1 | step 1600 | batch 1600/2250 | lr
0.00015 0.00001 | loss 0.4706 | s/batch 7.86
2025-12-09 14:39:18,276 INFO: | epoch   1 | step 1650 | batch 1650/2250 | lr
0.00015 0.00001 | loss 0.4895 | s/batch 8.18
2025-12-09 14:46:03,478 INFO: | epoch   1 | step 1700 | batch 1700/2250 | lr
0.00015 0.00001 | loss 0.5097 | s/batch 8.10
2025-12-09 14:52:35,177 INFO: | epoch   1 | step 1750 | batch 1750/2250 | lr
0.00015 0.00001 | loss 0.3603 | s/batch 7.83
2025-12-09 14:59:10,681 INFO: | epoch   1 | step 1800 | batch 1800/2250 | lr
0.00015 0.00001 | loss 0.5012 | s/batch 7.91
2025-12-09 15:06:00,676 INFO: | epoch   1 | step 1850 | batch 1850/2250 | lr
0.00015 0.00001 | loss 0.5120 | s/batch 8.20
2025-12-09 15:12:14,177 INFO: | epoch   1 | step 1900 | batch 1900/2250 | lr
0.00015 0.00001 | loss 0.4480 | s/batch 7.47
2025-12-09 15:18:53,786 INFO: | epoch   1 | step 1950 | batch 1950/2250 | lr
0.00015 0.00001 | loss 0.4135 | s/batch 7.99
2025-12-09 15:25:26,276 INFO: | epoch   1 | step 2000 | batch 2000/2250 | lr
0.00011 0.00001 | loss 0.3835 | s/batch 7.85
2025-12-09 15:33:30,879 INFO: | epoch   1 | step 2050 | batch 2050/2250 | lr
0.00011 0.00000 | loss 0.5122 | s/batch 9.69
2025-12-09 15:52:20,077 INFO: | epoch   1 | step 2100 | batch 2100/2250 | lr
0.00011 0.00000 | loss 0.4131 | s/batch 22.58
2025-12-09 15:59:32,076 INFO: | epoch   1 | step 2150 | batch 2150/2250 | lr
0.00011 0.00000 | loss 0.4908 | s/batch 8.64
2025-12-09 16:17:21,277 INFO: | epoch   1 | step 2200 | batch 2200/2250 | lr
0.00011 0.00000 | loss 0.4216 | s/batch 21.38
2025-12-09 16:38:52,877 INFO: | epoch   1 | step 2250 | batch 2250/2250 | lr
0.00011 0.00000 | loss 0.7644 | s/batch 25.83
2025-12-09 16:38:52,896 INFO: | epoch   1 | score (76.14, 67.14, 69.49) | f1
69.49 | loss 0.6940 | time 20546.61
2025-12-09 16:38:52,986 INFO:
              precision    recall  f1-score   support

          科技     0.8137    0.8568    0.8347      1697
          股票     0.8100    0.8881    0.8472      1680
          体育     0.9089    0.9445    0.9264      1405
          娱乐     0.8177    0.8733    0.8446       971
          时政     0.7527    0.7803    0.7663       710
          社会     0.7397    0.7079    0.7234       558
          教育     0.8529    0.8154    0.8337       455
```

```
          财经      0.7373    0.4167    0.5324       384
          家居      0.7175    0.6925    0.7048       374
          游戏      0.7562    0.6559    0.7025       279
          房产      0.7560    0.5826    0.6580       218
          时尚      0.7143    0.5743    0.6367       148
          彩票      0.7833    0.5875    0.6714        80
          星座      0.5000    0.0244    0.0465        41

    accuracy                           0.8114      9000

   macro avg      0.7614    0.6714    0.6949      9000
weighted avg      0.8071    0.8114    0.8051      9000
```

机器学习

这个跑出来的分类提交之后，分数只有0.06。

明明验证集的分数有0.74的，气死我了

```
# Count Vectors + RidgeClassifier
# 使用词袋模型（Count Vectors）结合岭回归分类器（RidgeClassifier）进行文本分类
# 由于数据做了匿名化处理，所以训练出来的东西似乎没有多大用处
# 可以通过词频分析给它逆向出来，原本的匿名化前的数据
# 这个跑出来的代码提交之后，分数只有0.06.。。明明验证集的分数有0.74的，气死我了
"""
One-hot
这里的One-hot与数据挖掘任务中的操作是一致的，即将每一个单词使用一个离散的向量表示。具体将每个字/词编码一个索引，然后根据索引进行赋值。

One-hot表示方法的例子如下：

句子1：我 爱 北 京 天 安 门
句子2：我 喜 欢 上 海
首先对所有句子的字进行索引，即将每个字确定一个编号：

{
    '我': 1, '爱': 2, '北': 3, '京': 4, '天': 5,
  '安': 6, '门': 7, '喜': 8, '欢': 9, '上': 10, '海': 11
}
在这里共包括11个字，因此每个字可以转换为一个11维度稀疏向量：

我：[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
爱：[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
...
海：[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
Bag of Words
Bag of Words（词袋表示），也称为Count Vectors，每个文档的字/词可以使用其出现次数来进行表示。

句子1：我 爱 北 京 天 安 门
句子2：我 喜 欢 上 海
直接统计每个字出现的次数，并进行赋值：

句子1：我 爱 北 京 天 安 门
```

```
转换为 [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]

句子2：我 喜 欢 上 海
转换为 [1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
在sklearn中可以直接CountVectorizer来实现这一步骤：

from sklearn.feature_extraction.text import CountVectorizer
corpus = [
    'This is the first document.',
    'This document is the second document.',
    'And this is the third one.',
    'Is this the first document?',
]
vectorizer = CountVectorizer()
vectorizer.fit_transform(corpus).toarray()
"""
import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import RidgeClassifier
from sklearn.metrics import f1_score
X = 10000
train_df = pd.read_csv(r'C:\Users\brighten\Downloads/train_set.csv', sep='\t',
nrows=15000)


vectorizer = CountVectorizer(max_features=3000)
train_test = vectorizer.fit_transform(train_df['text'])
clf = RidgeClassifier()
clf.fit(train_test[:X], train_df['label'].values[:X])

val_pred = clf.predict(train_test[X:])
print(type(val_pred))
print(f1_score(train_df['label'].values[X:], val_pred, average='macro'))


# 测试
test_df = pd.read_csv(r'C:\Users\brighten\Downloads/test_a.csv',sep='\t')
test = vectorizer.fit_transform(test_df['text'])
pred = (clf.predict(test))

submission = pd.DataFrame({
    'label': pred
})
submission.to_csv('submission_ridge.csv', index=False)
```

学习赛（第三季及以后）：**255/0.06**

学习赛（第三季及以后）

日期: 2025-12-08 21:04:29

分数: 0.0625

逆天