華中科技大學

项目开发报告

题目: 疫情期间网民情绪识别

课程名称: 机器学习
专业班级: CS1703
学　　号: U201714668
姓　　名: 葛松
指导教师: 李玉华
报告日期: 2020 年 7 月 25 日

计算机科学与技术学院

目录

1 疫情期间网民情绪识别

1.1 项目目的

1. 掌握文本处理的基本方法
2. 理解不同算法结果的差异性
3. 掌握对程序运行结果的评价方法

1.2 项目内容

1. 根据 train.csv 文件中的微博数据, 设计算法对 test.csv 文件中的 4500 条微博内容进行情绪识别, 判断微博内容是积极的 (1)、消极的 (-1) 还是中性的 (0)。
2. 通过混淆矩阵对算法的结果进行评价
3. 使用多种模型对问题进行处理，并进行对比

1.3 问题分析

所有的文本分析问题，基本都包括以下几个部分，下文将对这几个方面对问题进行分析

- 数据预处理，根据数据的特性进行清洗
- 数据转换，即将文本转化为可计算的的值
- 模型训练，将转换后的数据输入模型中进行训练，并对测试目标进行预测

1.3.1 数据预处理

由于本次项目的数据为微博真实数据，并且没有进行任何预处理，所以需要先对数据进行清洗以便后续工作。基本的预处理包括以下几个步骤

- 中文分词

  由于中文与英文不同, 不像英文一样自然以空格分割, 而是全部连接在一起的。所以需要对中文进行单词的分割。也即中文分词。以便以单词为单位进行后续的分析。中文分词在业内已经有比较成熟的解决方案，相关的工具包也比较丰富，如jieba分词, 清华大学的THULAC, 以及 MIT 的SnowNLP

- 去除停用词

  所谓停用词即为一些无法表征特别多信息的词语，如你, 我, 他，以及一些标点符号等等。这些词语出现的次数可能非常多，但是却无法带给我们有效的信息，反而会影响我们后续的计算，所以需要进行去除。在 github 上可以找到很多已经归纳好的停用词列表，可以通过综合多个停用词列表来得到一个更大的集合，从而增加去除的准确度。同时我们需要针对微博数据的特点进行针对性的处理，在给出的数据中很多情况下，一句话都包含有诸如 @账号名，又或是话题名等信息，如 //@新疆发布:# 众志成城打赢疫情防控阻击战 # 众志成城防疫情，我在新疆，我承诺! 中的 @新疆发布就是账号名，并不是评论本身。

- 去除频数较小的词 (可选)

  去除停用词之后可能还会有一些频数非常小的词语，其本身出现的次数不多，但是这类词语会极大地增加我们的词语集合，如果采用one-hot 编码，将会使我们的向量变得非常庞大，难以计算（本次实验中去除后的词语集合达到了 2 万多个）。幸运的是，在训练词向量的过程中，相关的函数基本上都会提供一个可选的参数来设置词语的最小出现次数，非常方便的解决了这个问题。

1.3.2 数据转换

所有的机器学习问题都要面临的一个问题就是如何将采集到的数据变为可以计算的数字，而针对文本处理，基本有以下几种常用的转换方法

- 独热编码 (one-hot)

  即将每一个句子转化为一个长度为词库大小的向量，当包含这个词语时就将对应位置设置为 1，否则为 0，但是由于词库往往非常大，会导致生成的数据非常庞大，并且非常稀疏，导致计算效果不佳

- 整数编码

  将每一个词语用唯一的数字标示进行标示，整个句子将表示为词语标识的序列，常常需要对句子的长度进行统一，常见于简单实现的神经网络模型中

- 词袋模型 (bag-of-word)

  与独热编码唯一的不同就是保存了出现次数，而不仅仅为 0,1 值

- 词嵌入 (word-embedding)

  通过CBOW或者Skip-Gram等模型，将每一个词语转化为指定维度的向量表示，生成出的向量则包含了该词语的语义信息。相近的词语在高维空间中所对应的距离往往也比较小。再通过句子所包含的所有词的词向量来获取该句子所对应的向量（通常为直接将每一个词语的向量相加）。相较于前面的几种方法，词嵌入有压缩语义信息、向量长度较短等优点。

而在本次项目中，为了提高准确率，分别针对普通线性模型，如svm、logistics回归，以及简单的神经网络，分别采用了词嵌入以及整数编码的方法，同时辅以PCA等降维方法，使得训练时间进一步减小。

1.3.3 模型训练以及评价

在实际运行过程中，实际上尝试了非常多种方法，如KNN、SVM、Logistic回归。亦或是直接调用他人写好的文本分析库如SnowNLP，以及一些神经网络等等。但是由于正确率等原因，最终还是选择了logistic回归，SVM以及简单的神经网络作为本报告介绍的对象。

对于最终效果的评价，采用了题目中所要求的三种评价指标，分别为

- Precision (精准率)
- Recall (召回率)
- F1-score (F1) 分数

分别对不同模型的预测结果给出相应的评价

1.4 设计与分析

该部分与问题分析部分一一对应，分别介绍不同阶段的设计与主体代码，整体程序流程图如下图所示

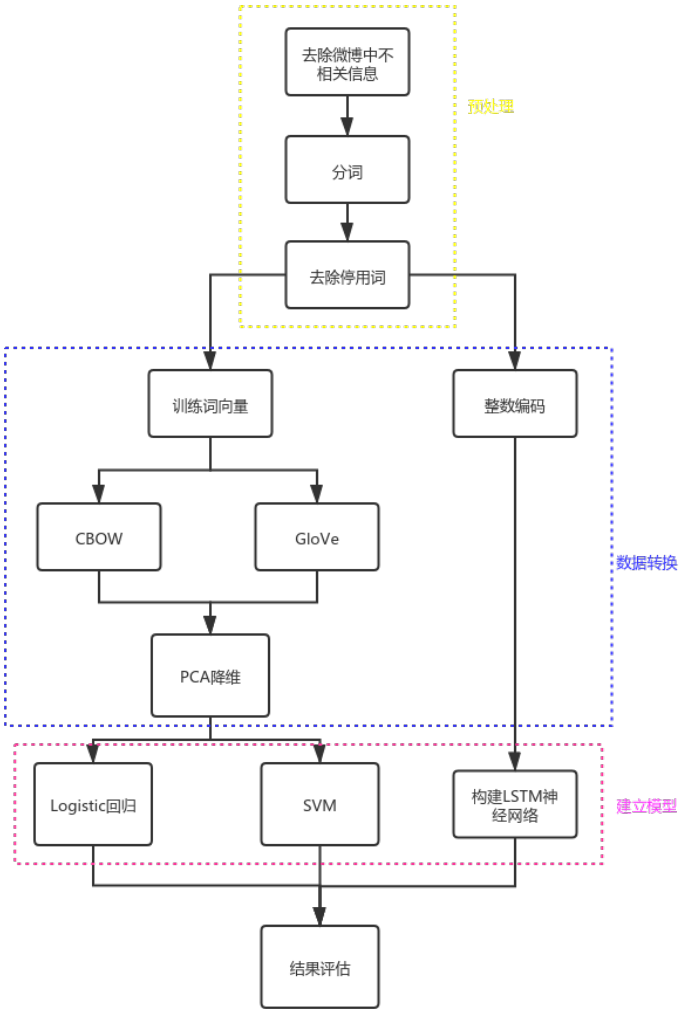


图 1: 数据清洗结果

1.4.1 数据预处理

分词

在本次实验中，我们采用业界比较成熟的分词工具jieba来对数据进行分词，通过以下函数可以非常方便的将中文句子分割为一个个的单词

```
words = jieba.lcut(sentence, HMM=False)
```

去除停用词

我们采用 github 上一个非常齐全的停用词列表仓库来构成本项目所用到的停用词表。包括哈工大停用词表，百度停用词表等等。同时定义以下函数来综合多个停用词表。

```
def makeStopWord():
    stopword = set()
    for file in listdir('stopwords'):
        if file.endswith("txt"):
            with open(f'stopwords/{file}') as f:
                lines = f.readlines()
                for line in lines:
                    words = jieba.lcut(line, cut_all=False)
                    for word in words:
                        stopword.add(word.strip())
    return stopword
```

去除账号名等不相关数据

观察微博数据中账号名以及话题出现的模式，发现可以通过正则匹配来进行去除，如下代码便使用了正则表达式来匹配多种可能的情况，去除不相关数据

```
def clean_redundant(comment):
    comment = re.sub('#.*?#', '', comment)
    comment = re.sub('//@.*?:', '', comment)
    comment = re.sub('//@.*?: ', '', comment)
    comment = re.sub('//.*?:', '', comment)
    comment = re.sub('//.*?: ', '', comment)
    comment = re.sub('【.*?】', '', comment)
    comment = re.sub('《.*?》', '', comment)
    comment = re.sub('//.*?//', '', comment)
    comment = re.sub('@.*?: ', '', comment)
    comment = re.sub('@.*?:', '', comment)
    comment = re.sub('『.*?』', '', comment)
    comment = re.sub(r'\d', '', comment)
    return comment
```

在经过分词、去除停用词、以及针对微博特性的清洗之后，原数据变为了易于处理的单词格式，如下图所示

```
[ ] train_df
```

	微博中文内容	情感倾向	jieba_cut
0	中国加油一起加油//@企鹅倩儿:#医护后盾#致敬所有前线医护工作者和志愿者们辛苦，因为你们我...	1	[中国, 加油, 加油, 致敬, 前线, 医护, 工作者, 志愿者, 辛苦, 平安, 加油,...
1	决定一个人最终高度的，往往并非起点，而是拐点，机遇都在拐点！2020年是鼠年，鼠在12生肖里...	1	[最终, 高度, 起点, 拐点, 机遇, 拐点, 年, 鼠年, 鼠, 生肖, 里, 排, 第...
2	我们都会尊重"遍体鳞伤但仍心无恨意，笑对人生的人"。陶勇医生，加油！	1	[尊重, 遍体鳞伤, 心, 恨意, 笑对人生, 陶, 勇, 医生, 加油]
3	【#关注新型冠状病毒疫情#7大热点，钟南山李兰娟院士答疑】现在疫情处于什么阶段？"方舱医院"...	1	[疫情, 处于, 阶段, 方舱, 医院, 收治, 一类, 患者, 双黄连, 热门, 药品, ...
4	解释:病毒变成结晶在空气里，然后人or别的呼吸进去了，它就寄存在别的生物体里复制它的RNA（...	0	[解释, 病毒, 结晶, 空气, 里, 呼吸, 寄存, 生物体, 里, 复制, RNA, 高...
...	...	...	...
44596	#你平安回来我承包一年家务#【疫情期间如何解决焦虑情绪？】人类的悲喜不相通，焦虑却出奇一致....	-1	[人类, 悲喜, 相通, 焦虑, 出奇, 害怕, 安静, 逃避, 独处, 陷入, 恶性循环,...
44597	#seu通知#日前，#新型冠状病毒#感染的肺炎疫情持续蔓延，江苏省教育系统已启动突发公共卫生...	0	[日前, 感染, 肺炎, 疫情, 持续, 蔓延, 江苏省, 教育, 系统, 启动, 突发, ...
44598	//@新疆发布:#众志成城打赢疫情防控阻击战#众志成城防疫情，我在新疆，我承诺！#疫情防控动态#	1	[众志成城, 防疫, 情, 新疆, 承诺]
44599	//@吾六一061://@暖白系野原小丸子:女将军//@沈泊舟123://@画长平://@楚...	0	[女将军, 女将军, 陈, 薇, 院士, 李, 兰, 娟, 专家, 国宝级, 人物, 配, ...
44600	由于临床需求量太大，口罩和防护服等都是一次性消耗物品，加上很多捐赠不一定符合医用0800套防...	-1	[临床, 需求量, 太, 口罩, 防护服, 一次性, 消耗, 物品, 捐赠, 符合, 医用,...

44601 rows × 3 columns

图 2: 数据清洗结果

需要注意的是，经过了上述处理之后，一些句子可能会变为空，针对这些句子也进行了去除处理，但好在这种情况非常少，不会有很大的影响。

1.4.2 数据转换

以下分别介绍项目中所用的词嵌入以及整数编码的方法

词嵌入

词嵌入有多种不同的模型，其中GloVe模型以及CBOW模型是效果比较好的两种模型。通过将分词好的数据输入到对应模型中进行训练可以得到针对该数据集合的词向量。本项目中两种模型均进行了尝试。

- CBOW模型

  CBOW模型可以通过word2vec包来非常方便的进行训练，如下述代码所示

```
from itertools import chain
from gensim.models import Word2Vec
from gensim.models.word2vec import LineSentence
corpus = [" ".join(x) for x in chain(train_df.jieba_cut, test_df.jieba_cut)]
# 将语料库写入文件
with open('corpus.txt' , 'w') as f:
for i in corpus:
    f.write(i + "\n")
# 开始训练
# 维度为800
model = Word2Vec(LineSentence('./corpus.txt'), size=1000, window=5, min_count=5, workers=8)
```

  得到model后便可以通过model[word]的方式来获取某一个词语所对应的词向量。而具体训练的效果可以通过查看某一个词语的最相近词来间接查看。我们以关心为例，查看其最近的几个词语，如下图所示 可见训练结果还是比较理想的

```
[34] model.most_similar('关心', topn=10)

  /usr/local/lib/python3.6/dist-packages/i
    """Entry point for launching an IPythc
  /usr/local/lib/python3.6/dist-packages/g
    if np.issubdtype(vec.dtype, np.int):
  [('大众', 0.8442174196243286),
   ('好人', 0.8329925537109375),
   ('给予', 0.8205270767211914),
   ('拜托', 0.8167148232460022),
   ('敬佩', 0.8123922944068909),
   ('尽职尽责', 0.8116296529769897),
   ('李医生', 0.8059046864509583),
   ('散播谣言', 0.8028981685638428),
   ('鼓励', 0.7999752759933472),
   ('记住', 0.7963657379150391)]
```

图 3: 与关心最近的词语

- GloVe模型

  与CBOW模型有标准 Python 实现不同，斯坦福的GloVe模型的官方实现为 C 语言，所以需要从 github 克隆其仓库，并修改 shell 脚本中的配置内容，再进行训练。脚本内容以及训练过程如下图所示

```
[ ] done.
    vector size: 800
    vocab size: 18751
    x_max: 10.000000
    alpha: 0.750000
    07/24/20 - 10:44.46AM, iter: 001, cost: 0.086631
    07/24/20 - 10:46.04AM, iter: 002, cost: 0.073231
    07/24/20 - 10:47.23AM, iter: 003, cost: 0.066984
    07/24/20 - 10:48.42AM, iter: 004, cost: 0.061197
    07/24/20 - 10:50.00AM, iter: 005, cost: 0.055665
    07/24/20 - 10:51.19AM, iter: 006, cost: 0.050593
    07/24/20 - 10:52.38AM, iter: 007, cost: 0.045903
    07/24/20 - 10:54.00AM, iter: 008, cost: 0.041630
    07/24/20 - 10:55.18AM, iter: 009, cost: 0.037747
    07/24/20 - 10:56.37AM, iter: 010, cost: 0.034233
    07/24/20 - 10:57.55AM, iter: 011, cost: 0.031051
    07/24/20 - 10:59.14AM, iter: 012, cost: 0.028148
    07/24/20 - 11:00.34AM, iter: 013, cost: 0.025513
    07/24/20 - 11:01.53AM, iter: 014, cost: 0.023130
    07/24/20 - 11:03.11AM, iter: 015, cost: 0.020993
    07/24/20 - 11:04.29AM, iter: 016, cost: 0.019079
    07/24/20 - 11:05.48AM, iter: 017, cost: 0.017365
    07/24/20 - 11:07.06AM, iter: 018, cost: 0.015832
    07/24/20 - 11:08.25AM, iter: 019, cost: 0.014458
    07/24/20 - 11:09.43AM, iter: 020, cost: 0.013227
    07/24/20 - 11:11.01AM, iter: 021, cost: 0.012129
    07/24/20 - 11:12.20AM, iter: 022, cost: 0.011150
    07/24/20 - 11:13.38AM, iter: 023, cost: 0.010286
    07/24/20 - 11:14.56AM, iter: 024, cost: 0.009528
    07/24/20 - 11:16.15AM, iter: 025, cost: 0.008863
```

```
demo.sh
15 #   rm text8.zip
16 # fi
17
18 CORPUS=../corpus.txt
19 VOCAB_FILE=vocab.txt
20 COOCCURRENCE_FILE=cooccurrence.bin
21 COOCCURRENCE_SHUF_FILE=cooccurrence.shuf.bin
22 BUILDDIR=build
23 SAVE_FILE=vectors
24 VERBOSE=2
25 MEMORY=8.0
26 VOCAB_MIN_COUNT=5
27 VECTOR_SIZE=800
28 MAX_ITER=30
29 WINDOW_SIZE=15
30 BINARY=2
31 NUM_THREADS=8
32 X_MAX=10
33 if hash python 2>/dev/null; then
34     PYTHON=python
35 else
36     PYTHON=python3
37 fi
38
39 echo
40 echo "$ $BUILDDIR/vocab_count -min-count $VOCAB
41 $BUILDDIR/vocab_count -min-count $VOCAB_MIN_COU
```

图 4: GloVe 训练过程

其中维数也设置为 800。同样以关心为例，查看其最近的几个词语，如下图所示 可见效果也还不错

```
[31] model.most_similar('关心', topn=10)

  /usr/local/lib/python3.6/dist-packages/
    if np.issubdtype(vec.dtype, np.int):
[('作假', 0.6822944283485413),
 ('无微不至', 0.5609314441680908),
 ('同情', 0.4643115997314453),
 ('提问', 0.45355653762817383),
 ('公众', 0.43980318307876587),
 ('回答', 0.41460320353507996),
 ('爱护', 0.3883034586906433),
 ('关爱', 0.3425981402397156),
 ('新闻自由', 0.32545173168182373),
 ('帮凶', 0.3153856098651886)]
```

图 5: 与关心最相近的词语

上述两个模型均可以在一定程度上实现词语的嵌入。但是得到的结果向量维度还是非常高。这里我们采用了 PCA 降维的方法来将其映射到低维空间，且尽量不损失信息。从而方面后续计算

首先需要计算不同维度下的信息损失程度，由此再确定要将其降维到多少，具体代码如下所示

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn import metrics

# 载入数据
df = data
y = df.iloc[:,0]
x = df.iloc[:,1:]
##原始维度为800
n_components = 800
pca = PCA(n_components=n_components)
pca.fit(x)
#打印 pca.explained_variance_ratio_

##PCA作图
plt.figure(1, figsize=(12, 8))
plt.clf()
plt.axes([.2, .2, .7, .7])
plt.plot(pca.explained_variance_, linewidth=2)
plt.axis('tight')
plt.xlabel('n_components')
plt.ylabel('explained_variance_')
plt.show()
```

结果如下图所示 可见在 200 维的时候，既达到了大幅降低了维度的目的，同时也基本没有损失信息。

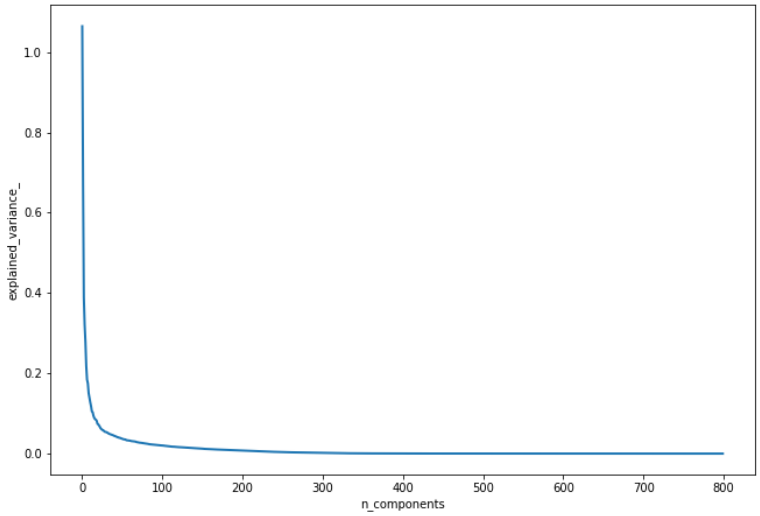


图 6: PCA 降维

整数编码

整数编码的方法在本项目中主要用于实现简单的 LSTM 神经网络，使用了`tensorflow`作为具体实现，而`tensorflow`也提供了相关的函数进行整数编码。

其中需要确定每一个句子的长度，不足的补 0，多余的部分裁去。通过对清洗后的句子的长度绘制出分布图，如下图所示

```
[ ]  sns.distplot(length, color="b")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd36375ac88>
```

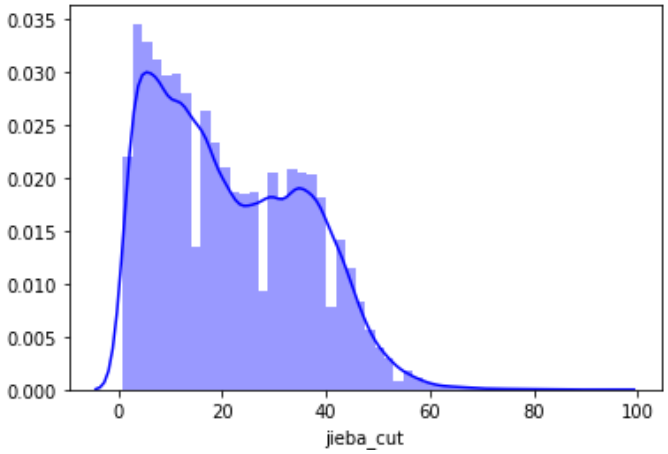


图 7: 句子长度分布

可以发现长度设置成 60 比较合适

1.4.3 模型训练

本报告选出了实际运行结果相对较好的几种模型进行介绍，分别有

- Logistic
- SVM
- LSTM网络

其中Logistic以及SVM这两种模型可以通过调用sklearn包中的默认实现，输入上文所提到的PCA降维后的数据进行训练即可。通过多次训练，调整参数找到较优解。具体代码如下所示

```
# 逻辑斯特回归
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=0.001, random_state=1, max_iter=10000)
lr.fit(X_train, y_train)
# 分别计算训练集以及测试集的正确率
lr.score(X_train, y_train)
lr.score(X_test, y_test)

# 支持向量机
from sklearn import svm
clf = svm.SVC(C=0.1, probability=True, cache_size=8192)
clf.fit(X_train, y_train)
# 分别计算训练集以及测试集的正确率
clf.score(X_train, y_train)
clf.score(X_test, y_test)
```

而LSTM模型通过tensorflow可以非常方便的实现，代码如下所示

```
## 两层LSTM，中间用Dropout层避免过拟合
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim),
    tf.keras.layers.LSTM(embedding_dim),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(embedding_dim, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(3, activation='softmax')
])
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
    metrics=['accuracy'])
## 遍历数据10次
num_epochs = 10
history = model.fit(X_train_padded,
                    train_df["情感倾向"]+1,
                    epochs=num_epochs,
                    validation_data=(X_validation_padded, test_df["情感倾向"]+1),
                    verbose=2)
```

1.5 结果分析

1.5.1 Logistic回归

使用word2vec作为词向量生成，降维后运行结果如下所示

使用LogisticRegression

```
[47] from sklearn.linear_model import LogisticRegression
     lr = LogisticRegression(C=0.001, random_state=1, max_iter=10000)
     lr.fit(X_train, y_train)
     lr.score(X_train, y_train)
```

```
0.6474294298334118
```

```
[48] lr.score(X_test, y_test)
```

```
0.5699238009861048
```

```
print(classification_report(y_test, lr.predict(X_test)))
```

```
              precision    recall  f1-score   support

          -1       0.65      0.73      0.69      1489
           0       0.52      0.44      0.48      1488
           1       0.52      0.54      0.53      1485

    accuracy                           0.57      4462
   macro avg       0.56      0.57      0.57      4462
weighted avg       0.56      0.57      0.57      4462
```

图 8: Logistic回归运行结果-word2vec

使用GloVe作为词向量生成，降维后运行结果如下所示

使用LogisticRegression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=0.001, random_state=1, max_iter=10000)
lr.fit(X_train, y_train)
lr.score(X_train, y_train)
```

```
0.6485953229748211
```

```
[59] lr.score(X_test, y_test)
```

```
0.5755266696548633
```

```
print(classification_report(y_test, lr.predict(X_test)))
```

```
              precision    recall  f1-score   support

          -1       0.65      0.74      0.69      1489
           0       0.54      0.46      0.49      1488
           1       0.53      0.53      0.53      1485

    accuracy                           0.58      4462
   macro avg       0.57      0.58      0.57      4462
weighted avg       0.57      0.58      0.57      4462
```

图 9: Logistic回归运行结果-GloVe

可见在相同配置下，GloVe生成的词向量比CBOW生成的词向量更加优秀，这也与业界实际情况相符。

1.5.2 SVM

使用word2vec作为词向量生成，降维后运行结果如下所示

```
[68] clf = svm.SVC(C=0.1, probability=True, cache_size=8192)
     clf.fit(X_train, y_train)
     clf.score(X_train, y_train)

     0.6780341247954081

[69] clf.score(X_test, y_test)

     0.557373375168086

[70] print(classification_report(y_test, clf.predict(X_test)))

                   precision    recall  f1-score   support

               -1       0.66      0.73      0.69      1489
                0       0.49      0.40      0.44      1488
                1       0.50      0.55      0.53      1485

         accuracy                           0.56      4462
        macro avg       0.55      0.56      0.55      4462
     weighted avg       0.55      0.56      0.55      4462
```

图 10: Logistic回归运行结果-word2vec

使用GloVe作为词向量生成，降维后运行结果如下所示

```
from sklearn import svm

[ ] clf = svm.SVC(C=0.1, probability=True, cache_size=8192)
    clf.fit(X_train, y_train)
    clf.score(X_train, y_train)

    0.6788412815856146

[ ] clf.score(X_test, y_test)

    0.5616315553563425

[ ] print(classification_report(y_test, clf.predict(X_test)))

                  precision    recall  f1-score   support

              -1       0.66      0.73      0.69      1489
               0       0.50      0.41      0.45      1488
               1       0.51      0.55      0.53      1485

        accuracy                           0.56      4462
       macro avg       0.56      0.56      0.56      4462
    weighted avg       0.56      0.56      0.56      4462
```

图 11: Logistic回归运行结果-GloVe

可见不管是SVM还是Logisitc，使用GloVe生成的词向量都要更加好。

1.5.3 LSTM

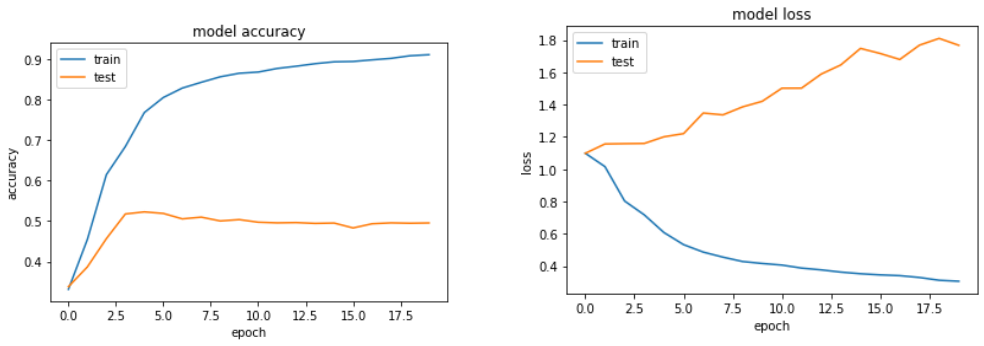
使用tensorflow构建简单的LSTM模型，将转换的整数编码数据输入，设置epoch为 20，进行训练，如下图所示

```
num_epochs = 20
history = model.fit(X_train_padded,
                    train_df["情感倾向"]+1,
                    epochs=num_epochs,
                    batch_size=1024,
                    validation_data=(X_validation_padded, test_df["情感倾向"]+1),
                    verbose=2)
```

```
Epoch 1/20
44/44 - 20s - loss: 1.0990 - accuracy: 0.3316 - val_loss: 1.0984 - val_accuracy: 0.3380
Epoch 2/20
44/44 - 19s - loss: 1.0160 - accuracy: 0.4550 - val_loss: 1.1565 - val_accuracy: 0.3870
Epoch 3/20
44/44 - 20s - loss: 0.8030 - accuracy: 0.6148 - val_loss: 1.1583 - val_accuracy: 0.4567
Epoch 4/20
44/44 - 20s - loss: 0.7172 - accuracy: 0.6843 - val_loss: 1.1596 - val_accuracy: 0.5177
Epoch 5/20
44/44 - 20s - loss: 0.6078 - accuracy: 0.7682 - val_loss: 1.2008 - val_accuracy: 0.5229
Epoch 6/20
44/44 - 20s - loss: 0.5327 - accuracy: 0.8054 - val_loss: 1.2205 - val_accuracy: 0.5190
Epoch 7/20
44/44 - 20s - loss: 0.4867 - accuracy: 0.8287 - val_loss: 1.3480 - val_accuracy: 0.5056
Epoch 8/20
44/44 - 20s - loss: 0.4554 - accuracy: 0.8429 - val_loss: 1.3365 - val_accuracy: 0.5099
Epoch 9/20
44/44 - 20s - loss: 0.4283 - accuracy: 0.8565 - val_loss: 1.3860 - val_accuracy: 0.5004
Epoch 10/20
```

图 12: LSTM训练过程.png

同时可以通过训练历史绘制出正确率与 loss 的变化曲线，如下图所示



(a) 模型正确率 (b) loss 变化

图 13: LSTM训练变化过程

可以发现在训练过程中，在第 5 个epoch时达到最佳状态，测试集正确率 0.52，训练集正确率 0.78，之后则发生了过拟合，训练集上的正确率不断增加但测试集的正确率却进入瓶颈并有减小的趋势，其 loss 也不断增加。

1.5.4 不同方法比较

分别列出不同模型或方法的正确率，如下表所示

模型	precision 均值	recall 均值	f1-score 均值
Logitstic+GloVe	0.57	0.58	0.57
Logitstic+word2vec	0.56	0.57	0.57
SVM+GloVe	0.56	0.57	0.57
SVM+word2vec	0.55	0.56	0.55
LSTM	0.50	0.52	0.50

表 1: 不同方法的指标对比

可见不同方法之间的差异不大，其中`Logitstic+GloVe`的方法效果最好，而神经网络的方法最差。由于前者使用了词嵌入的方法，而后者只是简单的整数编码，相较之下，后者损失了不少信息。

同时本人在实验过程中尝试将项目给出的训练集以及测试集进行重新乱序分割，其正确率可以达到 0.7 以上，所以有点怀疑这个实验的测试集是不是有问题。而题目中又明确指出需要使用提供的测试集，所以最终的效果不是非常好，基本都在 0.56 左右。并且本项目与常规的情绪二分类有所不同，采用的是三分类，所以就更加困难。

1.6 思考与总结

1.6.1 个人体会

经过本次项目，我对于文本处理有了更加深刻的理解。通过查阅各种资料，甚至是看 cs224n(著名自然语言处理课程)。了解了各种文本处理的方法，如数据预处理，各种不同的数据编码方式，以及各种词向量的算法，如`Skip-Gram`、`GloVe`等等。也尝试了各种其他的方法，如`tfi-df` 以及`bag-of-word`，其中有些方法效果不是非常理想所以就没有在报告中体现（可以在代码中找到）。虽然各种方法综合起来，最高的正确率只有 0.57，但是在整个学习的过程中，结合课程中学到的知识，以及实际工程上实现的过程，我对于各种模型的建立，以及各种可调节参数的调节方法都有了更深入的了解。

同时，对于在训练集上可以达到 90% 以上的正确率，而在测试集上却始终无法突破 60% 的情况，我也进行一些测试。发现通过混合测试集以及训练集，并重新随机分割，正确率可以达到 70% 以上。所以怀疑这个测试数据是不是有些问题。

1.6.2 建议与意见

个人感觉课程中学习到的知识，其实无法处理像项目一这种比较综合的项目，许多课程中学习到的模型都无法很好的在项目一中运用。当然也有其他的项目的难度与课程内容匹配，比如项目六。也许在提供此类比较” 超纲” 的项目的时候，可以提供一些相关的参考资料，可以很大程度上减少我们自己毫无方向的学习。其他的方面还是比较不错的提供了充足的时间让我们来做这些项目，老师在群里答疑也很及时。