

为微信聊天个性化回复表情包

杨巧文 17307130294

2021 年 1 月 17 日

目录

1	任务描述	1
2	相关工作	2
2.1	文本分类算法	2
2.2	和表情有关的 nlp 任务	2
3	数据集	4
4	生成句向量	5
5	分类器	6
5.1	Baseline	6
5.2	fastText	7
6	总结与不足	10
6.1	总结	10
6.2	不足	10
7	附录	11

1 任务描述

我的选题动机来源于老师上课说的，往年有同学做的任务是给一句话配上表情。于是我想到，和配 emoji 表情相比，如果做和表情包相关的任务会怎么样呢？

而且，表情包已经逐渐成为了微信聊天中的重头戏，成为了活跃气氛、表达感情的利器。随着大家收藏的表情包越来越多，表情包的选择成为了一个问题。我从自己的微信聊天记录出发，使用预训练的词向量和词袋模型生成句向量，并用 MLP 对句子进行分类，以求为一句话挑选一个恰当的表情包来回复。实验选取了我本人微信聊天中最常用的 10 个表情包，结合它的上一句话，构建了一个小数据集，并在上面进行了训练和测试。

过去已经有不少人做了 emoji 和文字相关的任务。然而和单纯的 emoji 相比，表情包能够承载的涵义无疑是更为丰富的。一个表情包可以应对的对话场景可能比起一个 emoji 要宽泛很多，因为 emoji 的涵义往往是固定的，而且它们的数量也是如此；而表情包由于可以原创，就有了更多可以发挥的空间，比如我自己最常用的表情如图1。因此，表情包的预测需要对文字的情感分析把握得更准。



图 1: 呆猫猫

为了简化这个复杂的问题，而且考虑到聊天记录涉及到个人隐私，很难大量获取，我的思路是仅使用我自己的微信聊天记录训练出一个分类模型，且仅选择我自己使用次数最多的十个表情。这样的话，这个模型虽然比较小，但是具有个性化，而且算法也可以推广到其他人的聊天记录上。

在这个基线模型的基础上，我考虑也可以对网络做更多改进。所以我想使用 fast-Text¹作为我进一步探索的模型。考虑到 fastText 的速度快、准确率也高，我认为使用它在我的小数据集上进行模型训练是比较合理的。

¹<https://fasttext.cc/>

2 相关工作

接下来，我将简要地介绍一下文本分类相关算法，以及和表情相关的 nlp 任务。

2.1 文本分类算法

文本分类算法²包括传统算法和深度学习算法。

传统的分类算法将任务拆分成特征表示和分类这两个部分。特征提取大致有文本预处理、特征提取、文本表示这几个步骤，分类可以使用诸如贝叶斯、KNN、决策树、SVM 这样的分类算法。

其中，文本预处理过程对于中文的自然语言处理来说，可以分成分词和剔除停用词这两个步骤。和英文相比，中文的特殊点在于需要分词，因为相比于单个的字，词语才是可以表意的最小语言单位。而分词算法又有多个种类，比如字符串匹配、统计等等。基于统计的分词算法比较丰富，诸如上课所说的隐马尔科夫模型、n-gram 等等。

而特征提取和文本表示，通常指的是将文本表示为计算机可以理解的格式，比如最常用的词向量方式。独热向量是最简单的词向量，它的向量维度就是词的数目；对于不同的词，在不同位置上置 1，其他维度上都是 0，就可以实现词的向量表示。但是这种表示方法过于稀疏，且维度可能过大。除此之外，还有词袋模型、向量空间模型等等。后来的词向量生成算法更多地考虑了如何压缩维度、以及在空间中反映词之间的距离关系等问题，因此准确度和可用性都更好。

相比于传统方法，深度学习方法常常能够用更低的人力来进行特征提取、文本表示这个步骤。效果较好的深度学习方法诸如 fastText（基于 MLP），TextCNN（基于 CNN），TextRNN（基于 RNN）等。它们的共同特点在于运用神经网络进行特征提取和分类任务，并且可以端到端地预测。

fastText 是我本次使用的一个网络结构，它的特征在后文中我会详细讲。TextCNN 的详细原理如图2。通过卷积神经网络强大的特征提取能力，以及使用类似 Google 的 Inception Module 结构来使用多个核来卷积，TextCNN 能够较好地完成文本特征提取的任务。

TextRNN 则是基于 n-gram 的思路，考虑到文本中词语之间的联系，于是使用了 RNN 来捕捉这种联结。而 RNN 的结构，如图3，更擅长处理时间序列。

2.2 和表情有关的 nlp 任务

比较有名的一个项目³是吴恩达的机器学习课上介绍的 Emojify，即通过一句话，来生成一个对应的 emoji 匹配它。Emojify 的网站是：<http://emojify.tk/>。

Emojify 的 v1 版本只使用了一个 softmax 分类器来做，如图4，可以对于小数据集做分类。v2 版本则引入了 LSTM，如图5，更能考虑到句中词之间的关系。

²<https://zhuanlan.zhihu.com/p/76003775>

³<https://zhuanlan.zhihu.com/p/47762573>

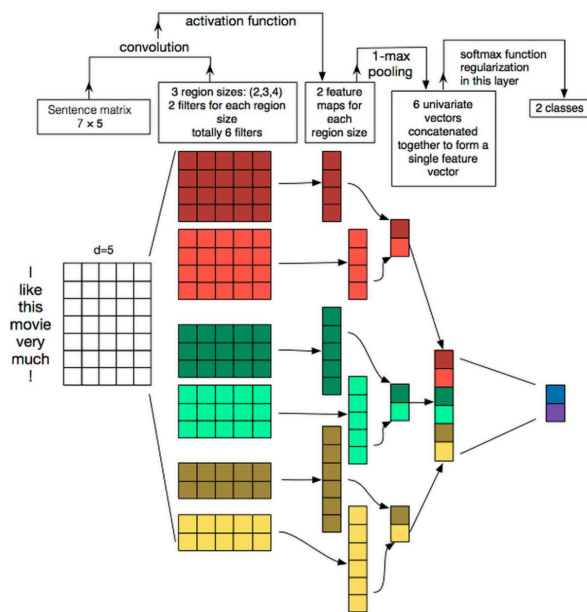


图 2: TextCNN 结构

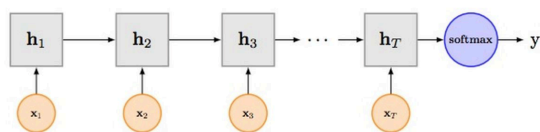
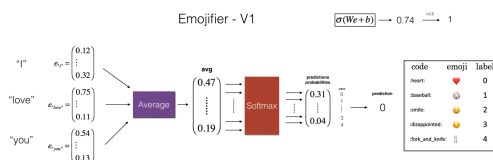
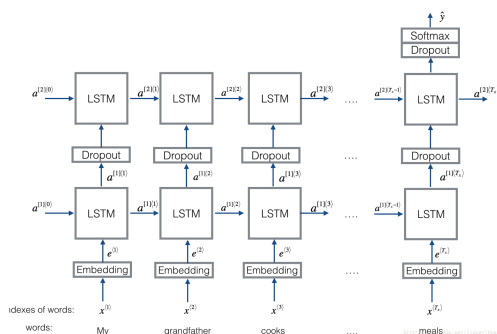


图 3: TextRNN 结构

图 4: Emojify-v1 结构⁴图 5: Emojify-v2 结构⁵

3 数据集

我在电脑上拿到了自己的微信聊天记录。经过查阅相关资料⁶，得知它们存储在 sqlite 的 database 里面。于是我将这些聊天记录导出到 csv 文件，并进行处理。

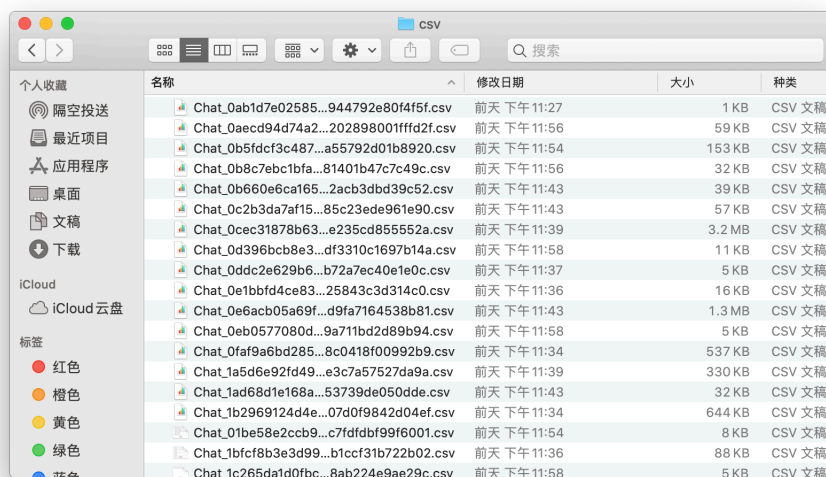


图 6: 导出的 csv 文件

接下来，就是找到聊天记录中的表情包。我通过查找资料⁷了解了微信聊天记录的格式。之后我提取出了所有聊天中的静态表情包（有些表情包是通过 md5 码发送的，未能成功提取到），并且拿到了这个表情包的前一句话，组成了一个 < 上一句话，表情包回复 > 的 pair。这样的 pair 总共有 3393 个。

之后，我提取出出现次数最高的十个表情，并将对应的 pair 拿到了，按照 8:2 的比例划分训练集和测试集。这十个表情如下图7：

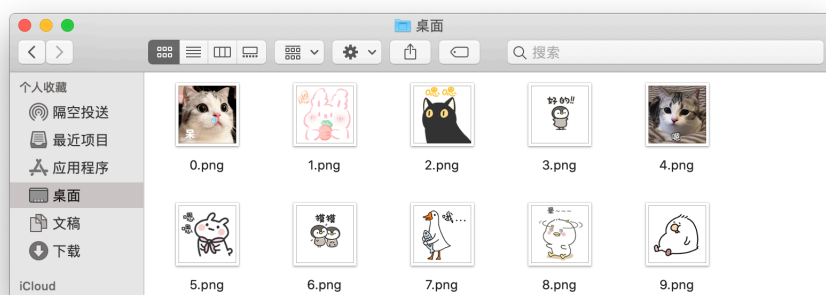


图 7: 最常用的 10 个表情

⁶<https://blog.csdn.net/swinfans/article/details/88712593>

⁷<https://github.com/nodeWechat/wechat4u>

4 生成句向量

句向量的生成有多种方法，我采用的是最简单的词袋模型，即忽略句子中各个词语之间和关系和权重，直接将句子内的词向量求平均并作为句向量。

词向量的生成我使用了腾讯 AI Lab 放出的预训练好的中文词向量⁸。由于词向量过大，我只使用了前 1,000,000 热度的词向量，大小有一个多 G，每个词维度是 200。在实践中，它们已经可以概括绝大多数词了。

腾讯词向量在以下几个方面作了改进⁹：

- 语料采集：语料不仅有新闻语料（腾讯新闻、天天快报），也有腾讯自己抓取的网页、小说等。这使得语料来源比较多样化，能够应对的场景也比较多。采用了新闻数据，使得语料库更加新鲜。
- 词库构建：引入了维基和百度百科的部分词条，还可以自动地从网页中发现新词，并计算它们的语义相似度。
- 训练算法：腾讯 AI Lab 自己研发的 Directional Skip-Gram 算法是词向量的训练算法。DSG 算法基于 Skip-Gram，并进一步将词对的相对位置引入考虑，从而增强了词向量的语义准确性。

拿到词向量后，我使用 jieba 对句子进行分词，然后平均词向量得到了句向量。过程大致如图8。

```
In [28]: demo = ["可以麻烦帮我烧一下水嘛", "nlp让我焦头烂额", "今天天气好好", "写一首诗给我看看", "能不能帮她写个代码"]

demo_seg = []
for sent in demo:
    seg_list = jieba.cut(sent)
    demo_seg.append(list(seg_list))
print(demo_seg)

[['可以', '麻烦', '帮', '我', '烧', '一下', '水嘛'], ['nlp', '让', '我', '焦头烂额'], ['今天天气', '好好'], ['写', '一首', '诗', '给', '我', '看看'], ['能', '不能', '帮', '她', '写个', '代码']]

In [29]: demo_model = KeyedVectors.load('1000000.bin')
w_size = demo_model.wv.syn0[0].shape[0]
demo_vec_pairs = []
for seg in demo_seg:
    sent_vec = [0] * w_size
    count = 0
    for w in seg:
        if w in demo_model.wv.vocab.keys():
            sent_vec += demo_model[w]
            count = count + 1
    if count != 0:
        for item in sent_vec:
            item /= count
        demo_vec_pairs.append(sent_vec)
print(demo_vec_pairs)

[array([ 1.88523801e+00, -1.60389701e+00,  3.60568002e-01,  1.64047502e+00,
        1.83092102e+00, -1.02719600e+00,  2.38900678e-03, -2.28459979e-02,
        1.44356697e+00,  2.88266978e-01, -1.72062006e-01, -8.31011009e-01,
        3.22810014e-01, -4.73012988e-01,  1.06846900e+00, -9.01253990e-01,
        -1.48342301e+00, -1.46204302e+00,  5.89166004e-01, -1.24624101e+00,
        1.93112998e-01,  2.73903993e-01,  1.44751501e+00,  7.46213015e-01,
        1.37797598e+00,  4.71738007e-01, -1.11739902e+00,  1.17341799e+00,
        2.45260301e+00, -6.32743994e-01,  1.41833996e+00,  2.59809006e-01,
        -1.20914017e-01,  1.81692303e+00, -9.35413007e-01, -8.60224032e-01,
        2.48404995e-01,  1.24196400e+00, -1.87909601e+00,  3.31139918e-02,
        1.15658499e+00,  4.67918988e-01,  2.60010197e+00,  7.05779985e-01,
        -5.48079006e-01, -9.19092984e-01, -1.62977500e+00, -2.20101598e+00,
        -6.08168986e-01, -2.05060039e-02, -1.91096098e+00, -4.47581016e-01,
        -7. 01277300e+00,  9. 25490000e-01,  1. 11551401e+00,  8. 40882022e-01,
```

图 8: 生成句向量

⁸<https://ai.tencent.com/ailab/nlp/en/embedding.html>

⁹<https://www.biaodianfu.com/tencent-word-embedding.html>

5 分类器

我的任务比较简单，就是通过 <上一句话，表情包回复> 这个 pair 训练一个端到端的分类器，判断对于一句话可以选择哪个表情包来回复。

5.1 Baseline

基线模型我采用的是多层感知器神经网络 (Multi-layer perceptron neural networks, 下简称 MLP), 如图9, 即一个输入层、一个隐层和一个输出层, 每个层上加 relu 激活函数。为了避免过拟合问题, 在每个层还加了 0.1 的 dropout。

和单个感知器相比, MLP 可以解决非线性问题, 比如 XOR 就不可以用一个感知器来解决, 但是可以用 MLP。多个感知器的组合, 就可以实现复杂空间的分割。

一个简单的 MLP 的 mapping function 可能是这样的:

$$y_k = f\left(\sum_{k=0}^{n_H} f\left(\sum_{j=0}^{d_2} f\left(\sum_{i=0}^{d_1} x_i w_{ji}\right) w_{kj}\right) w_{lk}\right) \quad (1)$$

其中, w 为各个层的参数, x 为输入, y_k 为输出; 在我的网络中, 输出是一个十维的向量, 每个位置代表选择每个类别的概率。

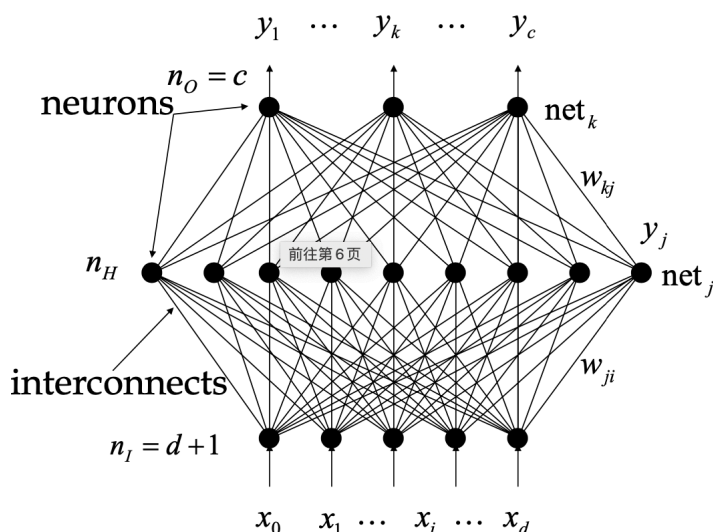


图 9: MLP¹⁰

训练过程中, 训练集和测试集的数据分别有 724 和 181 个。因为数据集比较小, 初始的 batchsize 定为 100, batch 数量定为 50, 学习率定为 0.01, Adam 优化的参数 weight_decay 定为 0.01, 并使用了交叉熵损失函数。结果如下:

这个结果可以说是确实不太行。初步判断为数据集太小导致过拟合。因为在超参上能做的努力我都已经尝试完了。为了进一步判断这个网络在实际样例中能否做出不错的表现, 接下来我又自己写了几句话让模型判断回复什么表情, 结果如图10:

epoch	train acc	val acc	loss
0	0.3273	0.3778	2.0966
10	0.3703	0.3833	1.7455
20	0.3786	0.3444	1.3530
30	0.4133	0.3611	1.2874
40	0.4424	0.2944	0.9952

表 1: Baseline 结果



图 10: 测试样例

发现尽管准确度不行，但是测试样例得到的回复大致都比较合理。猜想可能是聊天中语境比较复杂，比如有的表情回复的是之前的某句话，或者自己回复自己；以及比较重要的一点是，前十名热门的很多表情包表达的语义相近，比如“嗯”，“好的”之类，只是猫猫头不一样。这样的话，可能不少结果尽管判定是错的，但是实际上语义是可以表达的。

5.2 fastText

在完成了基线的实验之后，我尝试用 fastText 进行文本分类，从而进一步判断效果太差的原因是在网络上还是数据集或者句向量生成上。fastText 和其他分类方式相比有几个优点¹¹：

- 速度很快的同时，不会降低精度
- fastText 可以自己训练词向量，更加方便
- 优化：Hierarchical Softmax、N-gram

fastText 和 CBOW 比较相像，不同的是 fastText 预测的是 label，而 CBOW 预测的是中间词。fastText 的简单结构如图11。图中， x_1 到 x_N 表示文本中的 n-gram 向量，每个特征

¹¹https://blog.csdn.net/feilong_csdn/article/details/88655927

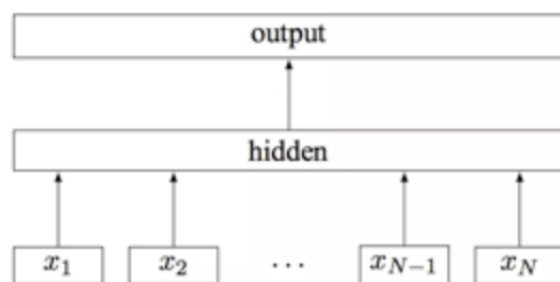


图 11: fastText 结构

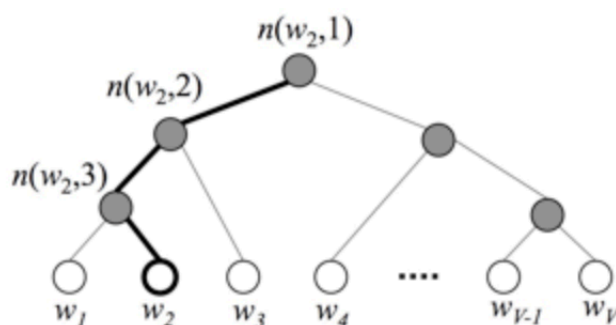


图 12: 层次 softmax 结构

是词向量的均值。fastText 会利用全部的 n -gram 去预测文本分类为每个 label 的概率，并在输出时使用了层次 softmax。

fastText 中使用的 Hierarchical Softmax 结构如图12。根据类别的频率构造霍夫曼树，可以将计算 softmax 时的复杂度从 $O(N)$ 降到 $O(\log N)$ 。通过计算非叶子节点处向左、向右的概率，层次地获取 softmax 值。比如，在某一节点，向左和向右的概率公式可能是这样的：

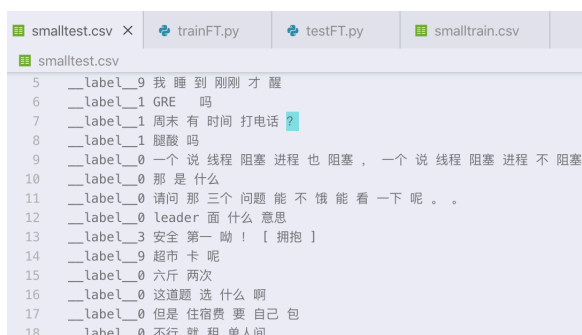
$$p(n, left) = \sigma(\theta_n^T \dot{h}) \quad (2)$$

$$p(n, right) = 1 - \sigma(\theta_n^T \dot{h}) \quad (3)$$

此外，fastText 还使用了 n -gram 算法。它的基本思想是将文本按照固定长度的滑动窗口进行划分，为每一次窗口移动所涵盖的 n 个词（或者字）生成词向量。使用它的好处在于可以更好地考虑到词之间顺序上的关系以及可以为罕见的字或者词生成向量。

实验使用了 fasttext 这个 python 包，训练集和测试集与 MLP 中相同，参考了来自https://blog.csdn.net/weixin_43977375/article/details/90200837的代码。为了符合 fasttext 的输入要求，对数据集做了一些调整，也预先用 jieba 进行了分词（如图13）。实验中测试了不同的超参，结果如表2。

可以看到，不管怎么调参，这个准确度和我自己训练的 MLP 相差无几，说明不是网



Index	Text	Label
5	__label__9 我睡到刚刚才醒	__label__9
6	__label__1 GRE 吗	__label__1
7	__label__1 周末有时间打电话	__label__1
8	__label__1 腿酸吗	__label__1
9	__label__0 一个说线程阻塞进程也阻塞，一个说线程阻塞进程不阻塞	__label__0
10	__label__0 那是什么	__label__0
11	__label__0 请问那三个问题能不饿能看一下呢。。	__label__0
12	__label__0 leader 面什么意思	__label__0
13	__label__3 安全第一 呦！[拥抱]	__label__3
14	__label__9 超市卡呢	__label__9
15	__label__0 六斤两次	__label__0
16	__label__0 这道题选什么啊	__label__0
17	__label__0 但是住宿费要自己包	__label__0
18	__label__0 不行就租单人房	__label__0

图 13: 符合 fastText 格式的数据集

accuracy	epoch	embedding dim	lr	wordNgrams
0.3646	1000	200	0.1	3
0.3646	500	200	0.1	3
0.3315	100	200	0.1	3
0.3646	1000	100	0.1	3
0.3591	1000	500	0.1	3
0.3536	1000	200	0.01	3
0.3923	1000	200	1	3
0.3756	1000	200	1	2
0.3646	1000	200	1	4

表 2: fastText 结果

络的问题，而是数据集的问题。我也用效果最好的一组参数测试了一些句子，输出的表情包 label 如图14。

```
(base) qiaowenyan@qiaowenangeMBP fastText % python3 testFT.py
Warning: 'load_model' does not return WordVectorModel or SupervisedModel any more, but a 'FastText' object which is very similar.
Building prefix dict from the default dictionary ...
Loading model from cache /var/folders/kv/sdwj_qkj3yq7221vf10mzbh80000gn/T/jieba.cache
Loading model cost 0.799 seconds.
Prefix dict has been built successfully.
可以麻烦帮我烧一下水嘛
4
nlp让我焦头烂额
0
今天天气好好
0
写一首诗给我看看
3
能不能帮她写个代码
0
```

图 14: fastText 预测样例

6 总结与不足

6.1 总结

本次实验，我选择了比较生活化、且有创意的课题，利用预训练词向量自己训练了文本分类器。

过程中遇到不少问题。比如说微信聊天记录的提取，以及网络过拟合等等。有些问题最后也没有解决，但是大体上完成了表情包回复 bot 的功能。

6.2 不足

目前我完成的部分还有以下不足：

- 数据集太少：可能可以通过微信的某些接口来将表情包的 md5 加密码转换成表情包，这样可以获取更多数据。
- 表情包情感分类：有不少表情包的涵义是相近或相同的，而这种表情包大量出现在我的微信对话中。因此可以考虑做一个表情包情感分类，从而让文本分类的结果能够对应的表情包范围更充分。
- 数据集不平均：数据集中各个类别的数据数量有很大差别，这也是导致网络效果差的一个重要原因。可以使用数据增强的方式来扩充数据，或者每个 batch 从各类别取同等数据进行训练。

主要都是数据方面的不足。

7 附录

附件的 `train.ipynb` 中有我全部的代码和训练过程。如果需要自己训练，可以先从<https://github.com/cliuxinxin/TX-WORD2VEC-SMALL> 按需下载词向量，按照我前面说的方法准备好自己的聊天记录，修改一下相关路径为自己电脑上的路径，然后就可以开始跑代码了。我自己的聊天记录由于涉及隐私，就不上传了，但是我将训练集和测试集的 < 词向量, label > 保存到了 `train_vec.txt` 和 `test_vec.txt`。如果想要跑的话可以加载它们。

对于 `fasttext` 的使用，可以查看 `trainFT.py` 和 `testFT.py`。由于模型太大就不上传了。可以参考我之前写的如何使用 `fasttext` 准备数据、进行训练然后得到结果。

也可以检查我的 `github` 仓库：https://github.com/QiaowenYoung/NLP_Meme_Chatbot。