基于神经网络的 QA 客服机器人设计文档

介绍

QA 客服机器人预先把问题和答案准备好存放在数据库或者内存中，如下表 1 所示。当用户询问时，在问题列表中搜索和用户语句最相似的那个问题，并取出其对应的答案返回给用户。

问题索引	问题	答案索引	答案
1	有什么保险	1	最近有个分红险优惠销售
1	有什么好的保险推荐	1	最近有个分红险优惠销售
1	帮我推荐一个保险	1	最近有个分红险优惠销售
1	有啥好的保险不	1	最近有个分红险优惠销售
2	理财保险怎么返现	2	保险到期，请联系客户经理办理返现
2	理财保险怎么返钱	2	保险到期，请联系客户经理办理返现
2	理财型保险怎么返还	2	保险到期，请联系客户经理办理返现
2	如何返还理财型保险	2	保险到期，请联系客户经理办理返现
2	请问理财型保险怎么拿到钱	2	保险到期，请联系客户经理办理返现
…	…	…	…
x	怎么查账户	x	请先安装 app
x	如何查询账户	x	请先安装 app
x	账户怎么查	x	请先安装 app
x	我要查账户	x	请先安装 app
x	app 查账户如何操作	x	请先安装 app

表 1 问题列表

现有的问题检索方案有关键字匹配的、基于规则的模型、基于统计的模型、基于神经网络的模型等。

基于关键字匹配：正则匹配

基于规则模型：构建语法树、正则匹配、词典匹配等。

基于统计模型：TF-IDF(SOLR)、SVM 等。

基于神经网络的模型有：textCNN、textRNN 等。

基于关键字、语法树以及词频权重的方式简单速度快，但是面对复杂的问题时，效果不理想，特别是用户用户问的问题超出预先准备的问题列表时，常常匹配不上，或者搜索不到最佳答案。基于神经网络的问题检索方案，由于提取了句子的更高维空间的特征，在泛化性能上比其他方案更优，结合知识图谱能达到理解更复杂的问题。

基于神经网络的 QA 模型通常有 2 种方案：

1. 基于分类模型的 QA
2. 基于距离向量的 QA

方案 1，基于分类模型的 QA

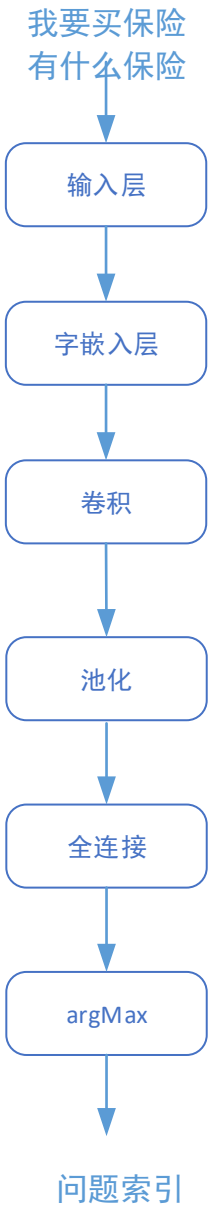


图1 基于分类模型的QA

有多少个问题就有多少个分类类别，预先用所有问题来训练模型，推理的时候用户的话经过模型的推理，得到一个问题索引，用索引到表 1 中取出相应的答案回复给用户。

优点：

网络简单，计算量少

缺点：

用户问了不相关的问题，比如关于保险的QA，用户说了完全不想管的问题，如“昨天在哪里玩的”，模型推理出来的置信度比较低，但是还是会有一个最接近的答案，什么时候该说听不懂，不太好判断。在这个模型解决这个问题的方案是增加集外类别，比如专门增加几个类别，准备一些不相关的问题或者话题，让神经网络学习出来，不相关的问题应该被识别成集外类别。

方案 2，基于距离向量的 QA

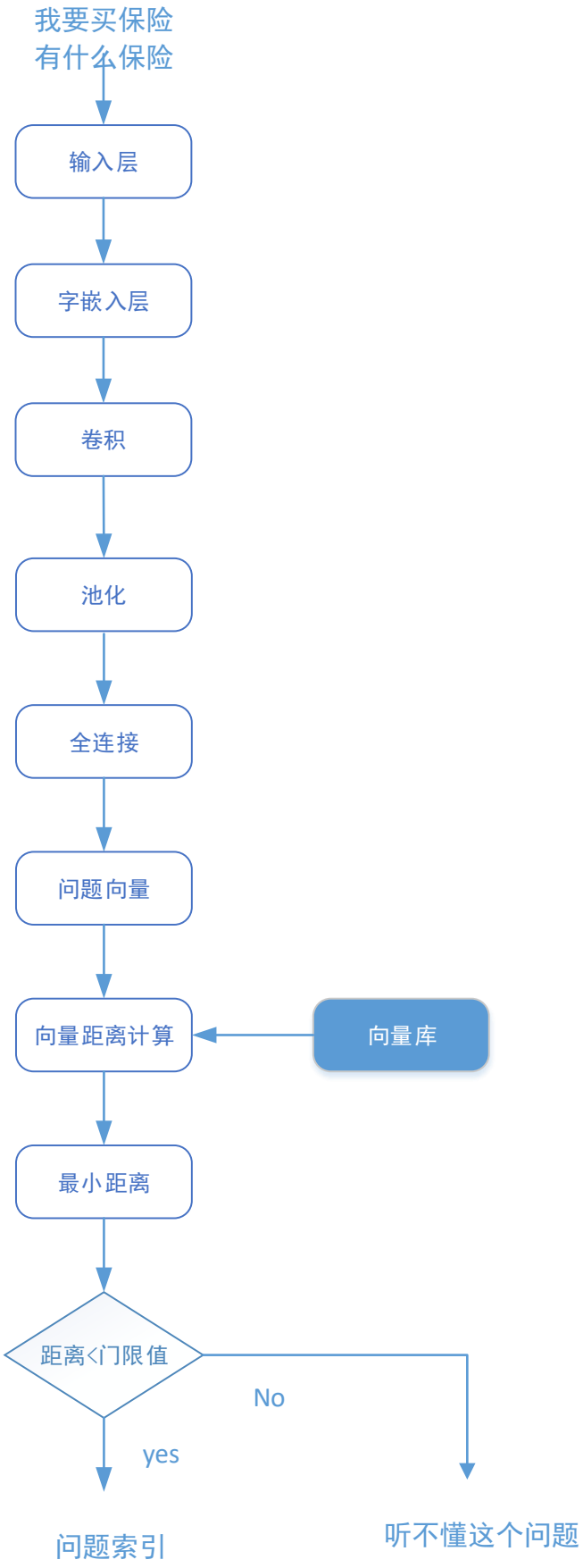


图2 基于距离向量的QA

预先用所有的类别的问题训练模型，并且把每一类问题的向量结果保存起来，同时计算出来一个最佳门限值，也就是说这个门限值使得正负测试样本的错误率最低。推理的时候每一句话得到一个高维向量，与预先保存的多个向量计算距离，通常是计算夹角余弦或者欧式距离，得到距离最近的一类问题就是结果，还需要把距离与门限值比较，小与门限值则认为用户这句话就是问的这个问题，否则认为用户说的话和预先设置的问题不相关，就可以提醒用户说“听不懂”。

优点：

由于存在向量距离门限值，不相关的问题容易辨别，F1 值相对比较高。

缺点：

每一句话都会得到一个高维向量，用这个向量去和预先存储的每一类问题的向量分别计算距离，计算量大，特别是问题比较多的时候。

架构图

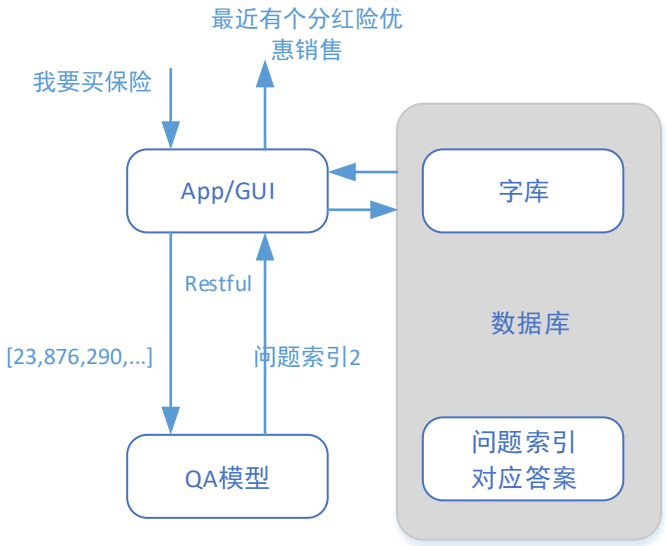


图3 QA模块图

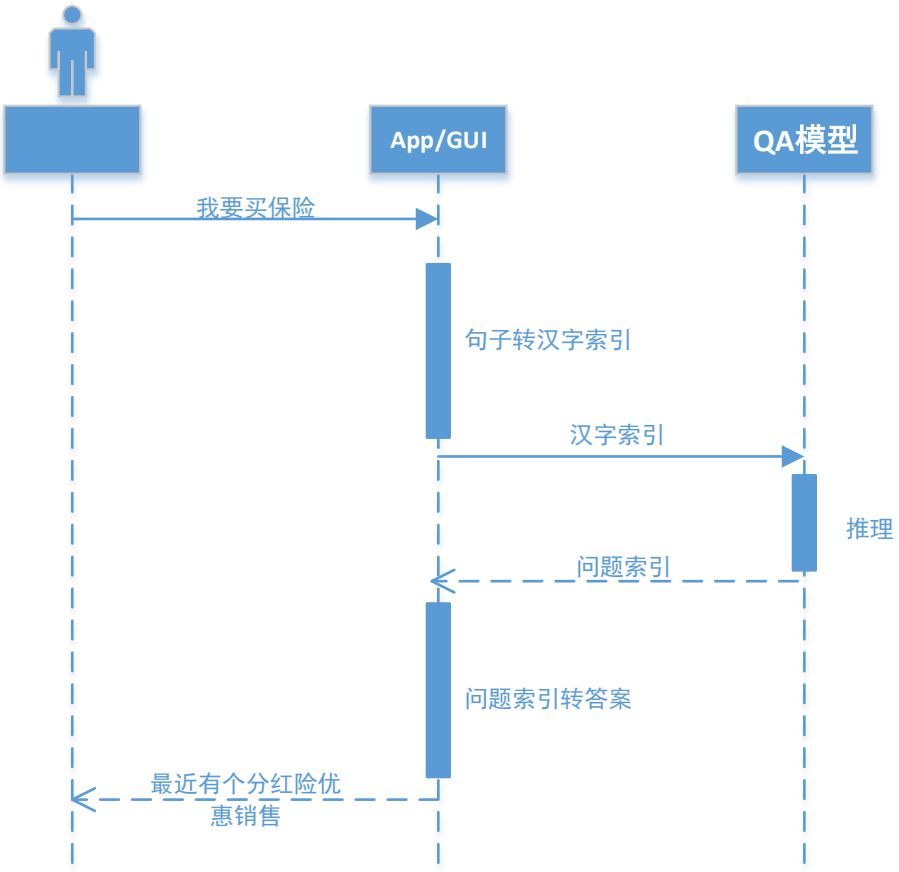


图4 QA时序图

App & QA 模型接口

本接口基于方案 2 来实现， QA 模型部署以后提供 restful 访问接口。模型给预先给 APP 提供字表，也就是汉字对应的索引数字。

Request:

Key	Type	direction
instances	array of int array	App to QA 模型

Example:

```
curl -X POST http://localhost:6009/v1/models/qa_model:predict -d {
"instances":
[[6888,398,2699,206,1435,2431,1090,329,2044,1006,251,6889,6887,6887,6
887,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,
6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887
,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,6887,688
7,6887,6887,6887,6887,6887,6887,6887,6887]]
}
```

Response:

Key	Type	direction
Predictions	array of float array	QA 模型 to App

Example:

```
{

    "predictions": [[0.38514185, 0.932220161, 0.920711517, -
0.156795338, -0.739072442, 1.75596642, -1.15974283,
0.386593521, 0.847169578, 0.526718438, 0.0194466412, -
0.726047158, 0.578923821, 0.112328663, -0.419979334, -
0.709555447, -0.278079867, -0.983806729, 0.320211589, 2.13562,
1.84979129, -1.14299417, 0.00881429, -1.34647381, 0.130547956,
```

```
        0.566979825, 1.29253197, -0.712399662, -0.185243726, -
        0.593760073, -2.36593795, -1.11128056]]
}
```

字表文件内容如下，每一行一个字，行号对应着索引号，app 收到用户的句子，通过字表把句子中的每一个汉字转成其对应的行号。

寻找向量距离最近的类别

模型预先算出每一个问题类别的均值向量，保存在文件中提供给 app，内容如下：

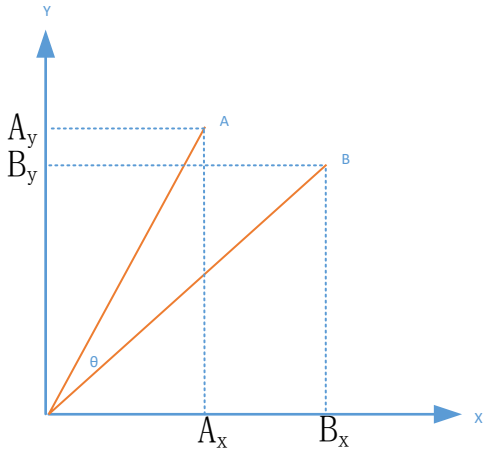
-3.669935645801680191e-01 -3.506616822310856287e-03 -5.664267880575997172e-01 4.223741271666118235e-01
-8.949425488710402998e-01 9.281406259536743031e-01 -1.555373680591583208e+00 -6.719974660873413441e-01
-1.679125252085695630e+00 -3.209213012965714262e-02 -9.627445263314330370e-01 1.171048223349275552e-01
-6.743744838237762940e-01 -1.482074394226074121e+00 -4.476945668458938432e-02 4.024871215224266052e-01
-1.155492178031376493e+00 -7.498005759857949426e-02 4.134363383054733276e-01 1.333914197626568043e+00 -
-2.707378019709879569e-01 -5.212634222225591696e-01 2.072520209240999445e-01 2.015067049312247338e+00 4
2.635037560911872578e-01 4.219449626494057304e-01 -4.484879631030408431e-02 -1.221977126749256071e+00 1
1.595812987237080982e+00 -4.831360004244059647e-01 -4.459530044972461749e-01 -1.073607237234602829e+00
8.799381769077287974e-01 6.208483041994247165e-01 9.385058811317337213e-01 -2.871544447951407952e-01 -2
-2.974965183685223113e-01 1.920768086488048310e-01 -4.157599630455175999e-01 3.931509931882222753e-01 -
-2.002570852637290955e+00 8.059590384364128113e-01 -8.510531261563301086e-01 -1.596212773583829403e-01
-2.349599332430580764e-02 8.135021785812311668e-02 2.821761246525456568e-01 -8.565353235980289748e-01 1
1.269434026696465212e+00 6.407090865752913533e-01 2.459588133814659960e-01 5.274537795815955610e-01 -8.
5.228278422728180885e-01 9.253116380423307419e-01 -7.707348559051752090e-02 -4.874628614634275436e-01 -

每一行代表一个问题的向量，行号代表问题索引，n 个问题得到 N 个向量，$V_1, V_2, \cdots, V_n$，QA 模型推理返回的是当前用户问题对应的向量 V_i，分别计算 V_i 和 $V_1, V_2, \cdots, V_n$ 之间的距离，找出距最小的向量的索引号，其对应的问题即用户期望询问的问题，找出该问题对应的答案返回给用户。

QA 模型预先提供一个门限值，如果得到的最小距离小于这个门限则认为问题搜索正常，如果距离值大于门限值，则认为用户问的问题在问题集中找不到，可以反馈用户换个问题或者听不懂。

假设 2 维的 2 个向量 A 和 B，计算它们的向量距离，本方案使用夹角余弦。

A 和 B 的夹角余弦：



$$\cos(\theta) = \frac{A_x B_x + A_y B_y}{\sqrt{A_x^2 + A_y^2} * \sqrt{B_x^2 + B_y^2}}$$

如果 A 和 B 是多维向量的夹角余弦值为：

$$\cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}}$$

计算距离的例子代码

Distance.h

Distance.cpp