Evaluating Rinna's Mind-reading Feature by Self-playing

Xianchao Wu[+] and Huang Hu[*]
[+] Microsoft Development Co., Ltd
Shinagawa Grand Central Tower, 2-16-3 Konan Minato-ku, Tokyo 108-0075
xiancwu@microsoft.com
[*] Graduate School of Software and Microelectronics, Peking University
No.5 Yiheyuan Road, Haidian District, Beijing 100871
tonyhu@pku.edu.cn

1 Introduction

In this paper, we introduce the detailed evaluation experiments for a simple "mind-reading" feature for an emotional chatbot, Rinna (Wu et al. 2016). This "mind-reading" feature works as the following way. First, the user is asked to think one topic word or a role (or, named entities, such as a famous personal name) and that topic word is unknown to Rinna (suppose Rinna has a large-scale role list yet Rinna does not know user's current choice). Then, Rinna tries to guess the topic word stored in user's mind by asking a sequence of questions, such as "Is your role a male or a female?". Based on user's responses such as "yes", "no", or "not sure", Rinna will consequently select the next question to ask until the candidate role is clear enough (such as the first candidate's probability is extremely larger than other candidates) or some termination conditions are met (such as a maximum number of questions of 20). In particular, the selection of the next question is based on user's answer to the former question and the existing prior ranking of the questions as well as mutex relations among questions. Utilizing no more than 20 questions (we also name this feature Q20), Rinna tackles user's mind by delivering the correct answer.

Figure 1 illustrates a real-world example for Rinna's Q20 mind-reading feature. Initially, user triggers this feature by typing "q20". Then, user is asked to think one role in his/her mind. When the user says "ready", Rinna starts to ask questions and receives user's answers of "yes", "no" or "not sure". The questions are ranked alike selecting a feature for splitting a note into subtrees in decision-tree algorithms (Quinlan, 1986), so that some target function is optimized. In our case, we first select questions that are at the "top" of the decision tree that is supposed to bring a maximum information improving by separating the roles into

User

q20

Think one famous people, Rinna can read your mind ☺

Ready!

Q1: Is your role virtual?

Yes

Q2: Is your role a male?

Yes

Q3: Is your role from Japanese Cartoon?

Yes

Q4: Is your role a student?

Yes

Q5: Is your role an athlete of sports balls?

No

Q6: Is your role's hair black?

No

Q7: Is your role's hair golden?

No

Q8: Is your role' hair brown?

Yes

Q9: Does your role have sisters or brothers?

Yes

Q10: Did your role kill people?

Yes

Q11: Is your role related to vampire?

No

Q12: Does your role drive robots?

Yes

Q13: Did your role go to the outer space?

Yes

Q14: Is your role with wavy hair?

No

Q15: Does your role have a sister?

Yes

Q16: Does your role revenge?

No

Q17: Does your role frequently ware white clothes?

No

Got that, your role is *Kira Yamato*, right?

Yes, Rinna, you are so smart!

Figure 1. An example for Rinna's Q20 mind-reading.

[*] Work done when Huang Hu was an internship student in Microsoft.

equally two groups. Also, the selection and user's answer of the former question determines the consequent ranking of the next question and the ranking of candidate roles. In Figure 1, Q1 has a keyword of "virtual", which influences Q3 having "Japanese Cartoon" with a quite close relation. Through receiving user's feedbacks of a list of keywords (underlines in Figure 1), the "key-value" attributes of one role is filled and the role is determined. This process is alike *jigsaw puzzles*, each question being taken as one piece of figure.

Besides employing decision tree ideas for ranking questions and roles, we can also take this as a gaming process between the user and Rinna. Our target is clear, to build a chatbot that is able to read users' mind with a high accuracy. The actions that Rinna can take is to select a next question from the candidate question list or show a candidate role from the candidate role list. The final reward for Rinna is user's feedbacks ("yes" or "no") for Rinna's final guessed roles. On the other hand, the actions for users include (1) selecting one role beforehand, (2) answering "yes", "no", or "not-sure" to Rinna, and (3) providing a final feedback to Rinna on her guess. Based on this analysis, we propose a self-playing pipeline by referring ideas included in (Silver et al., 2017) and (Sutton and Barto, 2018), with one "bot" acts as "questioning bot" and the other "bot" acts as "answering bot", for evaluating the accuracies of our Q20 feature using entropy-based question and role ranking algorithms (Wu et al., 2018).

2 Self-Playing Pipeline

```
Function1: selfPlaying(num) //num is the number for self playing
1.  Q20_feature = initialize(); //load data into memory, initialize the prior probabilities of questions and candidate roles;
2.  List<Role> refRoles = weightedRandom (Q20_feature.roles, num);
3.  foreach refRole in refRoles:
4.      onePlay(Q20_feature, refRole);

Function2: initialize()
1.  Q20_feature.roles = loadRoles(rolesFile);
2.  Q20_feature.questions = loadQuestions(qFile);
3.  Q20_feature.rightProbs = loadRightProbs(rFile);
4.  Q20_feature.qmutex = loadQMutex(qmFile);
5.  return Q20_feature;

Function3: weightedRandom(roles, num)
1.  Hashtable num2idx;
2.  pscores = 0;
3.  for (idx=0;idx<len(roles);idx++):
4.      pscores2 = pscores + roles[idx].pscore;
5.      for (i=pscores;i<pscores2;i++):
6.          num2idx[i] = idx;
7.      pscores=pscores2;
8.  outRoles = [];
9.  for (i=0; i<num; i++):
10.     anum = random(0, pscores) //a random number in the range of [0, pscores)
11.     outRoles.append(roles[num2idx[anum]]);
12. return outRoles;

Function4: onePlay(Q20_feature, refRole)
1.  session = [];
2.  for (i=0; i<20; i++):
3.      q = selectQuestion(Q20_feature.questions, session); //session assists "question bot"'s next question selection
4.      a = selectAnswer(refRole, q); //"answer bot" determines what to answer ("yes", "no", or "not sure")
5.      session.append([q,a]);
6.      if (max(softmax(Q2_feature.roles.pscore)) > 0.6):
7.          break;
8.  guessedRole = argmax_role{softmax(Q2_feature.roles.pscore)};
9.  if refRole == guessedRole:
10.     return "succeed";
11. else:
12.     return "failed";
```

Figure 2. Self-playing algorithm.

Figure 2 describes the self-playing algorithm we are using for Q20 accuracy and robustness evaluation. We lists four functions here in which (1) selfPlaying() is the major function, (2) initialize() takes the responsibility of loading necessary files to the memory, such as the list of roles, questions, the probabilities of users' historical selections and the mutex relations among questions, (3) weightedRandom() samples out the reference roles (which are only known to the "answer bot" and blind to the "question bot", i.e., Rinna) based on prior degrees of popularities, and (4) onePlay() that simulates one session of game playing by alternatively calling the "question bot" for selecting the next question and then calling the "answer bot" for selecting the (reference) answer to answer that question. From the "answer bot"'s point of view, the reference role is known so that that answer can be a mixture distribution of the reference answer with a random noise to testify the "robustness" of the intelligence of Rinna. When the number of questions exceeds 20 or when one candidate role's normalized probability (softmax function is used for normalization, Line 6 in onePlay()) is larger than a threshold, we stop the session and returns the guessed result. When the guessed result is identical with the reference role, the simulation succeeds.

Briefly, the data structure for our Q20 feature includes four tables, (1) the "Role" with names such as "Kira Yamato" (Figure 1), prior scores (pscore) obtained from search engine (such as Bing), and reference answers for each question; (2) the "Question" table with question names such as "Is your rule virtual?" (Figure 1); (3) the "rightProbs" table that stores users' selections of "yes", "no", and "not sure" for one question i for one role j; and (4) the "Mutex" table that stores relations alike "yes-yes" of two questions, such as "Is your role from Japanese Cartoon?" and "Is your role virtual?". (Wu et al., 2018) describes the detailed definitions of these four tables.

The selectQuestion() method used in Figure 2 is exactly the entropy-based question ranking algorithm as described in Equation (1) in (Wu et al., 2018) for computing weight q_i. We skip the details here. We mainly focus on how to add noise to the "answer bot". Figure 3 gives the answer selection algorithm with noisy allowed. The noisy here is basically to simulate those users' not sure answers since "not sure" brings no information gain for helping to rank the next question or current candidate roles. Extremely, we can also return "no" (or "not sure") when the reference answer is "yes" with a noisy probability. These two noisy can be easily integrated.

Function5: *selectAnswer*(refRole, q) //q is a question to be answered

1. rightP = refRole[q, 'yes']; //the probability that q is answered 'yes' in "rightProbs" and by referring reference answer;
2. wrongP = refRole[q, 'no']; //the probability that q is answered 'no' in "rightProbs" and by referring the reference answer;
3. notsureP = 1 – rightP – wrongP;
4. noise = 0.1; //or other values, such as 0.05, 0.2 and so on
5. maxP = max(rightP, wrongP, notsureP);
6. return (maxP == rightP) ? (rand() > noise ? 'yes':'not sure'/'no'): (maxP == wrongP)?(rand() > noise? 'no':'not sure'/'yes'):'not sure';

Figure 3. select answer algorithm.

3 Experiments

In our initial model, we selected Chinese as our test language and collected 10,633 famous people and virtual characters (such as from Japanese Cartoons) all around the world. We collected 1,800 questions. During a year-period of playing, we collected 27.5 million times of plays in which we obtained a top-1 prediction accuracy of 67.3%, in which 47.4% required 20 turns and the other 52.6% could terminate in an early stop. We first start our self-playing evaluation by adding no noisy information to obtain an upper bound of the accuracy and session length (number of questions asked by the "question bot" to conclude the role).

Figure 4 depicts the accuracy curve for a 100,000-time simulation of self-playing without adding noise to the reference answers. From the figure we can observe that the accuracy tends to convergence at around (0.954, 0.956) after 5,000 times. After that, the diversity of the accuracy tends to be trivial. Also, note that the final accuracy at the range of 0.955 is impressive and proves the effectiveness of the self-playing framework for Q20 measuring. Furthermore, the accuracy already reaches 0.954 at 3,000-time simulation, 0.949 at 2,000-time simulation, and 0.947 at 1,000-time simulation. All these results show the steady of the model under reference answers. In terms of speed, we can finish around 5,000 times of self-playing in one hour.

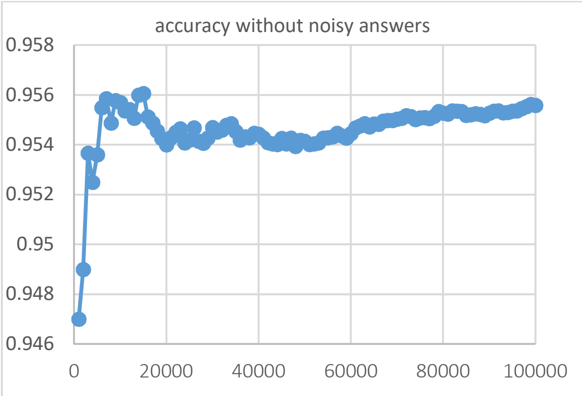

Figure 4. 100,000-time simulation accuracy.

session length	wrong	ratio	correct	ratio
9	3	0.1%	0	0.0%
10	11	0.2%	29	0.0%
11	35	0.8%	162	0.2%
12	53	1.2%	564	0.6%
13	52	1.2%	1681	1.8%
14	69	1.6%	3472	3.6%
15	50	1.1%	5663	5.9%
16	97	2.2%	8285	8.7%
17	135	3.0%	10544	11.0%
18	86	1.9%	11159	11.7%
19	24	0.5%	11108	11.6%
20	3826	86.2%	42892	44.9%

Table 1. Session length for the 100K self-playing.

Table 1 lists the session length for the correct and wrong simulations. For the correct side, 44.9% sessions need 20 questions to finish and the remaining 55.1% can terminate in an early stage. On the other hand, wrong predictions mostly (86.2%) terminates after costing 20 questions and only 13.8% stopped before that. We also manually analyze the wrong cases and compare the predicted roles with the reference roles. For almost all the cases, we found that there are close relations between the guessed roles and the reference roles, such as “Jinping Xi” (chairman of China) for the reference role and “politician” for the guessed role. We believe that the 20 questions could not distinguish these roles due to their internal close relations. One possible solution to this problem is to introduce novel questions from separable keywords of these roles for a clearer distinguishing.

noise	Accuracy1	Accuracy2	avg. wrong answer in "succeed" (left=Accuracy1, right=Accuracy2)		avg. wrong answer in "fail"	
0	0.951	0.957	0	0	0	0
0.05	0.821	0.593	0.8	0.5	1.5	1.4
0.1	0.641	0.298	1.5	0.9	2.4	2.3
0.15	0.471	0.147	2.1	1.4	3.4	3.1
0.2	0.313	0.069	2.8	1.8	4.2	3.9
0.25	0.200	0.030	3.4	2.3	5.1	4.8

Table 2. Accuracies of self-playing with noise, with a total self-playing account of 5,000 for each noise configuration.

Table 2 lists the accuracies of self-playing with noises taking values of from 0 to 0.25 (Figure 3) of changing from sure reference answers of “yes” or “no” to “not sure” (Accuracy1) and switching “yes” with “no” (Accuracy2) to stop transferring information to the “question bot” side. From the table, we observe that the accuracies drop significantly as the noisy answers are included more and more, yielding accuracies of from 0.951 to 0.200 (Accuracy1) and of from 0.957 to 0.030 (Accuracy2). Specially, we compute the average wrong answers respectively in the “succeed” and “fail” sessions, knowing that when there are averagely 0.8 to 1.5 wrong answers, Accuracy1 is at a level of 0.821, and when there are averagely 1.5 to 2.4 wrong answers, Accuracy1 is at around 0.641. On the other hand, Accuracy2 deteriorates even faster. Averagely 0.5 to 1.4 wrong answers will cause an accuracy drop of from 0.957 to 0.593. These accuracies (0.641 in Accuracy1 and 0.593 in Accuracy2) are the closest to the real-world user playing accuracy of 67.3% with 27.5 million sessions. Easy to see that there are nearly 2 questions were answered “not sure” which were supposed to obtain a clear “yes” or “no” answer. This also supplies a way of figuring out the too rare questions to avoid users’ answering of “not sure” in the real-world Q20 feature of Rinna. Extremely, when 5 of the 20 questions are answered “not sure”, the accuracy of the “question bot” is at 0.200.

4 Conclusion

We have presented a simple role-oriented mind-reading feature for our chatbot, Rinna (Wu et al., 2016), with million-level users. We achieved a top-1 accuracy of 67.3% under 27.5 million time playing. We describe the self-playing algorithm in this paper to estimate the accuracies of the “question bot” according to the changes of the noise information (for simulating users’ wrong answers). We report results that are helpful for further improving the robustness of our “mind-reading” feature in the future. We believe that our proposed ideas (together with the details described in (Wu et al., 2018)) and solutions are helpful for (1) improving the interestingness of real-world people’s interaction with virtual chatbots, and (2) extending novel pipelines of constructing user profiles by dynamically selecting related questions for determining which products they really want in scenarios of product recommendation.

References

Silver, David; Schrittwieser, Julian; Simonyan, Karen; Antonoglou, Ioannis; Huang, Aja; Guez, Arthur; Hubert, Thomas; Baker, Lucas; Lai, Matthew; Bolton, Adrian; Chen, Yutian; Lillicrap, Timothy; Fan, Hui; Sifre, Laurent; Driessche, George van den; Graepel, Thore; Hassabis, Demis. Mastering the game of Go without human knowledge. 2017. Nature. 550 (7676): 354–359.

Quinlan, J. R., (1986). Induction of Decision Trees. Machine Learning 1: 81-106, Kluwer Academic Publishers.

Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. 2018. The MIT Press.

Xianchao Wu, Huang Hu, Momo Klyen, Kyohei Tomita, Zhan Chen. Q20: Rinna Riddles Your Mind by Asking 20 Questions. 言語処理学会 2018.

Xianchao Wu, Kazushige Ito, Katsuya Iida, Kazuna Tsuboi, Momo Klyen. りんな：女子高生人工知能. 言語処理学会 2016.