Q20: Rinna Riddles Your Mind by Asking 20 Questions

Xianchao Wu[+], Huang Hu[*], Momo Klyen[+], Kyohei Tomita[+], Zhan Chen[+]
[+] Microsoft Development Co., Ltd
Shinagawa Grand Central Tower, 2-16-3 Konan Minato-ku, Tokyo 108-0075
{xiancwu, momokl, ktomita, zhanc}@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 a novel puzzle riddling framework for an emotional chatbot, Rinna (Wu et al. 2016). 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. Then, Rinna tries to guess (or, mind-reading) user's topic word 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", the candidate roles are ranked from the Rinna side. Sequentially, 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. With no more than 20 questions, Rinna is supposed to tackle user's mind by delivering the correct answer.

If Rinna's answer is confirmed by the user, then that gaming session is recorded for a future usage of data mining (such as updating the reference answers of one role's questions, in case that the candidate role's questions are answered differently with the reference answer). On the other hand, if Rinna could not provide a correct answer (i.e., all Rinna's answers were wrong), then Rinna asks the user to provide the correct answer and that session will be recorded as well for (1) new role appending and (2) existing role updating to Rinna.

We argue that this Q20 gaming benefits the following applications: (1) user profile construction, means that user's answer in mind reflects his/her interested topic, and (2) product recommendation, means that the roles can be replaced from famous person's names to partner companies' product names that allow Rinna to guess by following a similar Q20 workflow.

It is not trivial for constructing such a Q20 framework, due to the following difficulties: (1) a database of roles, or topic words of various domains (such as persons, foods, animals, places) should be constructed from Rinna's side; (2) respectively for each of the roles and topic words, a list of candidate questions together with their reference answers should be collected and constructed in a pre-determined form; (3) use users' historical answer recoding to update the reference answers to target roles' (such as the answer of one famous person's marriage related question changes from "single" to "married"). The background consideration is to ask million level users to *teach* Rinna about the "updating" of thousands of questions for million level roles and topic words, and (4) relationships among questions such that if one questions is answered by the user, then consequently the number of available questions and the ranking of them are dynamically tuned. For example, if a question (a) "Is your role a *male*?" is sent to the user and the user answers "yes", then questions such as (b) "Is your role a *female*?", (c) "Did your role obtain the best *actress* award of Oscar?" should not be asked anymore for that game session. Their answers are assigned to be "no" by default. Then, we define there is a "yes-no" relation between question (a) and questions (b), (c).

In this paper, we first describe the database structures for our Q20 framework. Then, we describe our role-ranking and question-ranking algorithms. Third, we describe a question relation classifier that utilizes neural networks and rich features. The classifier takes two questions as inputs and then predict five types of relations. We conclude this paper by showing statistics of launching this Q20 framework to Rinna family chatbots.

2 Q20's Data Structures

We first define the basic data structures for "Role", "Question", "Mutex" and "RightProb". Here, "Role" is a table that stores roles (such as personal names, product names and so on) and includes fields of:

1. Rid: unique role identity which is also the primary key of this table;
2. Name: name of the role, such as "George Washington";
3. Alias: alias of "Name", such as "1st US president";
4. Ans: a list of reference answers for the set of questions, such as "yes" for a question alike "Was your role the president of US?";
5. Rscore: prior score of current role, the score can come from long time query frequencies of "Name" and "Alias" of current role. The background consideration is that, one role will be ranked relatively high if its retrieval frequency is relatively high.

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

"Question" is a table that manages the structures of questions to be answered by the users of Rinna:

1. Qid: unique question identity, which is also the primary key of this table;
2. Qtext: question text, such as "Is your role a male?";
3. Alias: alias question list for "QText", such as "Is your role a man?"; With this alias, we can ask the users an identical question by different textual sentences;
4. tagKey: the tag of current question, such as "gender" for "Is your role a male?";
5. tagValue: the value of the tag, such as "male" for "gender" of current question. Note that, tagKey and tagValue are used to "generate" questions from given "attribute-value" of named entities which are mined from Wikipedia and related channels;

We also determine the mutex relations of questions by referring tagKey and tagValue in existing antonym and synonym lexicons. We define five types of relations between a pair of questions, namely, "yes-yes", "yes-no", "no-yes", "no-no", and "notsure". In order to describe the relationship between each question pairs, we use a table named "Mutex" which contains fields of:

1. Qid: one question id;
2. Yesyes: stores a list of qids that share a "yes-yes" relation with "qid". Such as there is a "yes-yes" relation between "Is your role born in Tokyo?" and "is your role born in Japan?";
3. Yesno: stores a list of qids that share a "yes-no" relation with "qid". Such as of between "Is your role a male?" and "Is your role a female?";
4. Noyes: stores a list of qids that share a "no-yes" relation with "qid". Such as of between "Is your role a male?" and "Is your role a female?";
5. Nono: stores a list of qids that share a "no-no" relation with "qid". Such as of between "Is your role virtual?" and "Does your role live in cartoon?".
6. Notsure: stores a list of qids that share a "not-sure" relation with "qid". Such as of between "Is your role a male?" and "Is your role virtual?". This relation is rather ambiguous and majorly influences the candidate set of next question(s). Basically, all questions that do not share the former four relations can be included in this "notsure" list.

It is essential to collect users' opinions about questions of roles. Especially, when Rinna successfully figures out a user's role in mind, then that user's answers to the 20 (or less) questions are important for (1) reference answer correcting, (2) the next session question/role ranking. For example, for one role's one question, suppose we did not assign its reference answer beforehand, and after a period, 80% of users answered "yes" to that question, then we can consider updating the reference answer of that question of from "not sure" to "yes". Motivated by this, we define a table named "RightProb" that has fields of:

1. Rid: one role identity;
2. Qid: one question identity;
3. *c*(*yes*), *c*(*no*), *c*(*not sure*): the three frequencies (counts) that users answered "yes"/"no"/"not sure" for "Qid" knowing that the final correct answer is "Rid". This frequency is collected when a Q20 game session success.

3 Question Ranking and Role Ranking

In this section, we respectively describe the ranking algorithms for candidate question ranking and for candidate role ranking.

The general framework of our algorithm is alike:

1. Select a candidate question *i* based on weight q_i;
2. Receive user's answer and update weight p_j for candidate roles;
3. If 20 rounds or early stop condition is satisfied, then goto 4; otherwise goto 1;
4. Show the top-1 role to the user, ask the user if it is correct; continue to show the top-3 roles if the user say wrong, ask the user again of the correctness;
5. Store current playing session.

In the following, we respectively explain how to compute q_i and p_j.

$$q_i = \sum_{j=1}^{J}\{p_j M_{i,j}\} + N(Y_i) + X_i. \quad (1)$$

Here, q_i stands for the weight of the *i*-th candidate question, p_j is initialized as the normalized score (probability) of the *j*-th candidate role,

$$p_j = \frac{Rscore(j)}{\sum_{j=1}^{J} Rscore(j)}, \quad (2)$$

in which Rscore(*j*) stands for the prior weight (such as yearly retrieving frequency of the *j*-th candidate role in Bing), and $M_{i,j}$ is the negative Shannon entropy for the binary Bernoulli distribution of the users' answers (in which, we only use "yes" and "no") of *i*-th question for *j*-th candidate role. Suppose there are *J* roles in total in database "Role".

$$M_{i,j} = Y_{i,j} log_2 Y_{i,j} + (1 - Y_{i,j}) log_2 (1 - Y_{i,j}),$$

$$Y_{i,j} = \frac{c(yes,i,j) + \sigma * \theta(Ans(i,j),' yes')}{c(yes,i,j) + c(no,i,j) + \alpha + \sigma * \theta(Ans(i,j),' yes')}.$$

Here, *c*(*yes/no*, *i*, *j*) is the frequency that users answer "*yes*" or "*no*" to the *i*-th question for the *j*-th role, $Y_{i,j}$ is the "*yes*" probability of the *i*-th question for the *j*-th role. We use α (e.g., α=1) for smoothing. Also, we introduce another parameter σ (e.g., σ=1,000) in case that the reference answer Ans(*i*,*j*) in database "Role" is '*yes*'. θ function here returns 1 when Ans(*i*, *j*) equals to '*yes*' and 0 otherwise.

The introduction of $\sigma * \theta(Ans(i,j),' yes')$ is to balance the "reference answer" and the "users' selection tendencies" for one question aiming for one role guessing. Note that, (1) the reference answer influences the ranking of the next question for collecting users' selections and (2) users' selection tendencies can further help updating the reference answers.

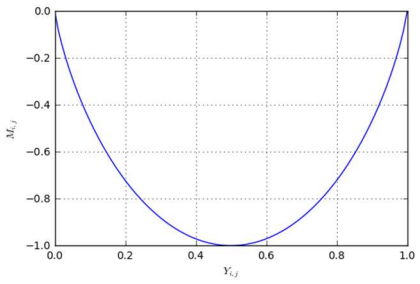

Figure 1. Shape of $M_{i,j}$ with respect to $Y_{i,j}$.

Figure 1 depicts the shape of $M_{i,j}$ being a function of $Y_{i,j}$. From the figure we see that we prefer to assign higher weights to those questions that have clear "*yes*" or "*no*" answers to all (or, as many as possible) candidate roles, due to $M_{i,j}$ takes higher values when $Y_{i,j}$ is closer to 0.0 or 1.0. These questions have rather higher distinguish abilities compared with questions that have "not-sure" answers where $Y_{i,j}$ is near 0.5.

In Equation (1), we use $N(Y_i)$, another Shannon entropy for the distribution of the expectation of Y_i under entity importance distribution p_j in which j ranges over from 1 to J. That is,

$$N(Y_i) = -\mathbb{E}(Y_i)log_2\mathbb{E}(Y_i) - \mathbb{E}(1-Y_i)log_2\mathbb{E}(1-Y_i)$$

Here,

$$\mathbb{E}(Y_i) = \sum_{j=1}^{J}\{p_j Y_{i,j}\}.$$

This item assigns a bonus for the questions that have a relatively larger "information" or "surprise degree".

The third item we use in Equation (1) is X_i for measuring the mutex information. X_i is computed during the interactive playing with one user. When selecting the first question, X_i is 0 for all candidate questions. Then, we use 1-order Markov constraint to compute X_i when selecting the next question, that is:

1. $X_i = X_i$ -1000, when i is the former question i';
2. $X_i = X_i$ -λ*{($p_0(i, i')+p_1(i, i')$)*$\theta(i',$ "*yes*")+($p_2(i, i')+p_3(i, i')$)*$\theta(i',$ "*no*")+$p_5(i, i')$*$\theta(i',$ "*not sure*")}, when the former question i' is answered "yes", we use p_0 and p_1 which measure (i, i')'s relations of "yes-yes" and "yes-no"; or when i' is answered "no", we use p_2 and p_3 which measure (i, i')'s relations of "no-yes" and "no-no"; or when i' is answered "not sure", we use p_4 which measures (i, i') "not sure" relation. Function $\theta(i',$ "*yes*") takes a value of 1 when the answer for question i' is "*yes*" and 0 otherwise. λ (such as 5, 10) here is a tunable hyperparameter.

Through X_i, we can punish q_i for those questions that share a strong relation (not "not-sure") with former questions and avoid asking the user to answer questions such as "is your role a woman?" right after a question alike "is your role a man?" being answered.

Finally, in order to bring more "random" to the game, we randomly select the next question from the top-N (such as 10) candidate questions ranked by q_i.

Now, we look at candidate role ranking. At the beginning, we use Rscore(j) for initializing p_j for the first-time ranking. Then, when one question i is answered, we update p_j by:

1. $p_j = p_j * Y'_{i,j}$ when i is answered "yes";
2. $p_j = p_j * N'_{i,j}$ when i is answered "no";
3. $p_j = p_j * (1-Y'_{i,j}-N'_{i,j})$ when i is answered "not sure".

Here, $Y'_{i,j}$ and $N'_{i,j}$ are defined as:

$$Y'_{i,j} = \frac{c(yes,i,j)+\sigma*\theta(Ans(i,j),'yes')}{c(yes/no/notsure,i,j)+\beta+\sigma*\theta(Ans(i,j),'yes')};$$

$$N'_{i,j} = \frac{c(no,i,j)+\sigma*\theta(Ans(i,j),'yes')}{c(yes/no/notsure,i,j)+\beta+\sigma*\theta(Ans(i,j),'yes')}.$$

Note that, we allow early stopping (also refer to Figure 2) by setting several conditions, such as when the first candidate role's p_j is 100 times larger than the second candidate role, or the value of normalized p_j is larger than 0.8. β (e.g., β=2) is for smoothing.

4 Question Relation Classification

When we compute X_i in Equation (1) or when we add a new candidate role to the database "Role" and manually assign the reference answers or when we append a new question, we will need to consider to automatically predict and utilize mutex relations among questions for time saving. We model this as a multiple-class classification problem, by taking a pair of questions as inputs and predict their relationships, each relationship being attached with a probability.

Making use of existing antonym and synonym lexicons, we manually construct a dataset with 38,375 samples and each sample is in a format of <q_1, q_2, relation> in which relation takes five possible values.

We train a multiple layer perceptron (MLP) neural network. The network contains one hidden layer with a dimension of 100, an input layer with 1,134 dimensions for representing bag-of-words of the two questions and 10 features for describing the similarities of the two questions, and finally a softmax output layer for predicting five classes.

These 10 features include, BM25 score (Robertson and Zaragoza, 2009), word2vec distance (averaged value) (Mikolov et al., 2013), edit distances of word and character levels, longest common sub-sequences of word and character levels, ratios of common words and characters, and string kernels (Lodhi et al., 2002) of word and character levels.

label	0 yes-yes	1 yes-no	2 no-yes	3 no-no	4 not-sure
#	1,885	25,645	1,623	390	8,832

Table 1. Dataset for classifier training.

Predicted	0	1	2	3	4	Recall
0	739	8	83	15	73	0.805
1	15	4,040	3	11	968	0.802
2	127	4	322	77	314	0.382
3	13	3	17	296	82	0.720
4	137	216	416	145	3,532	0.794
Precision	0.717	0.946	0.383	0.544	0.711	
Accuracy (micro-avg): 0.766						
Accuracy (macro-avg): 0.701						

Table 2. Confusion table for the classification results.

Q: is your role the King of some Cartoon?					
Q_1: is your role strange?					
p_{01234}	**0.267**	0.099	0.186	0.180	**0.268**
Q_2: is your role a King?					
p_{01234}	**0.265**	0.098	0.192	0.200	0.244

Table 3. Examples of question relations.

Table 2 shows the confusion table of the classification results of the five labels, together with class-specific precision and recall. We obtain accuracies of 76.6% under micro-average and 70.1% under macro-average. From this table, we see that the accuracies are largely influenced by the amount of training data (such as labels 2 and 3). Table 3 shows one question and two related questions with five relation probabilities attached. For these two questions, p_0 is relatively larger means a strong "yes-yes" relation between Q and Q_1 and between Q and Q_2.

5 Ask Users to Teach Rinna

Every time our chatbot successfully predicts the role in the user's mind, we store that game session into our related databases and timely update our ranking models, such as the answer counts c("*yes*", "*no*", and "*not sure*", i, j) for a given question i for a role j. Through this way, we can continuously learn the "correct" answer for one question related to one role. Suppose we assigned a "not sure" to "is your role married?" for a person A (such as a famous movie star). Then, after a period, more users answered "yes" to this same question for A, we can consequently set a condition to update the reference answer of from "not sure" to "yes" for A. Also, if A divorced after a period, and we can further update "yes" to "no" based on users' feedbacks. Generally, it is easy to extend the "yes", "no", "not sure" answers to natural language sentences or words such as "quite sure about that" (for "yes"), "never heard that" (for "no") or "maybe" (for "not sure"). Furthermore, the questions and answers can be designed to be domain specific, such as an answer "spicy" or "sweet" for "Which taste do you prefer?" of food domain.

6 Experiments

In our initial model, we selected Chinese as our test language and collected 10,633 famous people and virtual characters 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% and a top-3 prediction accuracy of 88.4%. Users also input the entity names (which are not covered by our database) for 6.4% of the playing sessions and 431K unique entity names were input by the users. This is a valuable source for appending novel entity names and for updating our related databases.

Figure 2 also depicts the distribution of number of turns for the 67.3% successful sessions, in which 47.4% required 20 turns and the other 52.6% could terminate in an early stop. This is important for saving

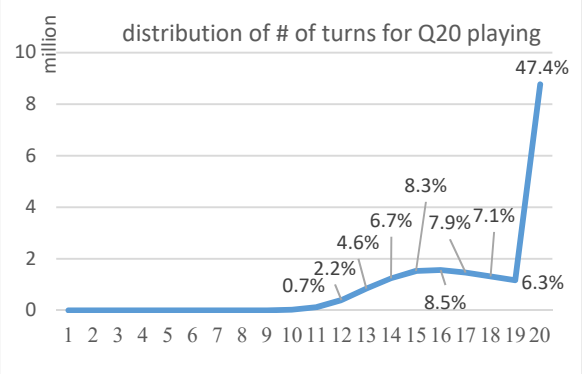

Figure 2. Distribution of num. of successful turns.

users' time for avoiding too many questions and ensure that their minds are detected successfully.

7 Conclusion

We have presented a simple role-oriented mind-reading feature for our chatbot, Rinna (Wu et al., 2016), with million-level users. We introduced the database structures of managing roles, questions, question relations, and users' answers for the questions of the roles. We designed entropy-based question ranking and role ranking algorithms by utilizing users' answer distributions, prior weights of the roles, and mutex relations among the questions for filtering next question selection. We also proposed a multi-class classifier making use of MLP for automatically determine the probabilities of relations between two questions. We finally achieved a top-1 accuracy of 67.3% and a top-3 accuracy of 88.4% under 27.5 million time playing.

We believe that our proposed idea 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. In the future, we would like to update the entropy-based ranking algorithms by deep reinforcement learning algorithms (Sutton and Barto, 2018). That is, by dynamically learning the action (question selection, candidate role sorting) ranking policies.

References

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

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text Classification using String Kernels. In JMLR 2002. Pp. 419-444.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In NIPS 2013.

Stephen Robertson and Hugo Zaragoza.The Probabilistic Relevance Framework: BM25 and Beyond. In Foundations and Trends in Information Retrieval archive. 3(4), 2009. Pages 333-389.

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.