PM Question Processing Final Project: 20 Questions

Wellesley Boboc
Universität Potsdam
Matriculation number: xxxxxx
email@domain

Anna-Janina Goecke
Universität Potsdam
Matriculation number: xxxxxx
email@domain

Rodrigo Lopez Portillo Alcocer
Universität Potsdam
Matriculation number: xxxxxx
email@domain

Elizabeth Pankratz
Universität Potsdam
Matriculation number: 804865
pankratz1@uni-potsdam.de

Abstract

lorem ipsum

1 Introduction: Task and motivation

Our task is to build a system that will play the game 20 Questions (20Q). A human player will be able to think of a target object, and the system will strategically select questions that allow it to narrow down the candidate objects in its knowledge base. It will incorporate the answers it receives and ultimately make a guess about what that target object could be. If the target object that the human player has in mind is not already in its knowledge base, the system will add it in based on the information the user has provided.

This task is interesting and challenging because it not only involves generating natural-language questions to present to the human player, but also choosing which questions are the best ones to ask, and manipulating the knowledge representation in accordance with the answers that the human player provides.

So, on the one hand, the project contains the computational-linguistic subtask of question generation (given a feature in the dataset, generate a natural-language question asking about that feature to display to the user), and on the other, the engineering subtask of knowledge base manipulation. For the latter, we will need to strategically select the best question to ask, incorporate the answers from the user as the game is played, and at the end, add previously-unseen objects into the knowledge base.

2 Related work

We will briefly explore two areas in the literature that are relevant for our implementation: existing approaches to implementing 20Q, followed by rule-based question generation (QG).

2.1 Previous 20Q implementations

Previous approaches to the implementation of a 20Q system have made use of diverse methods including probabilistic models (Dey et al., 2019), Reinforcement Learning (RL; Hu et al. 2018), and variations of Artificial Neural Networks (ANNs; Reddy et al. 2017; Burgener 2006; Tonin et al. 2018). The knowledge that the 20Q system has is often represented in a knowledge graph (KG; Dey et al. 2019), though some more sophisticated approaches also manage without (e.g. Hu et al., 2018).[1] Here, we will briefly discuss the relative merits of these implementations and how they inform our work on this project.

We begin with some probabilistic approaches. In general, these are characterised by maintaining a probability distribution over the set of outcomes. Consequently, none of the possible outcomes are ever totally discounted or thrown away—just associated with a lower probability. This approach is good for situations in which the questions are answered by the users in a "noisy" way (e.g. when users answer inconsistently or wrongly). In those cases, the system is still able to choose what the correct target object might be, even though the user has answered in a way that might seem incompatible with that object.

Dey et al. (2019) implemented a probabilistic model which operates on the dataset as weighted edge-node relations of a KG and updates throughout the course of the game. Here, the main idea of adjusting probabilities at every time step was exploited to generate a model that is able to predict the correct target object in fewer than twenty questions. Of particular interest is the way the

[1] A KG is essentially a graph where the nodes are entities and the edges between them are facts that connect them. For instance, a node *Macron* might be connected to a node *Paris* by the edge *lives in* (example from Godin et al., 2019), representing knowledge of the fact *Macron lives in Paris*.

model handles incorrect answers from the human player: the question generator does not fully reject or accept a certain object as being the target after every answer. f Instead, it rebalances the probabilities at each step in the game. To identify the target object, the model categorizes the questions into two layers, a primary layer (wide range of objects) and a secondary layer (specific range, targeted towards a small set of objects). Even though the model has been proven to perform very well, i.e. half of the target objects could be identified in fewer than ten questions, their work is very limited in that it is designed to only apply to Bollywood movies. However, the use of KGs to create 20Q (probably adapted from graph format into a tabular database; see below) is an approach that we would like to pursue for our 20Q implementation, by further elaborating the ideas of Dey et al. (2019), among others.

Hu et al. (2018) also rely on a probability distribution over all objects which is then updated according to the answers. However, their approach is different in that they use an RL framework: they implement a policy-based system of 20Q that uses reinforcement throughout the game. Instead of using a KG, the model for selecting questions is based on RL procedures trying to find the optimal reward function. Hu et al. (2018) suggest a neural network that they call RewardNet which learns the immediate reward at each time step to improve the overall performance of the model, since only receiving a reward at the end of the game wouldn't allow the system to learn for each question. The model continually improved its win rate over time and was shown to be able to identify the target object within 14 questions. While this system clearly has excellent performance, incorporating RL into our project is not feasible; it is included in this literature review only for the sake of completeness.

Another approach that is probably too sophisticated for us, but nevertheless useful to know about, is the use of neural networks. For example, Reddy et al. (2017) propose the application of KGs to generate sets of question-answer pairs within a Recurrent Neural Network architecture by deriving triple relations from given entities. The triples are composed of a subject, an object (both represented as nodes in the KG), and a predicate (represented as an edge in the KG). The model consists of two units: the *Question Keywords and Answer Extractor*, which directly selects necessary information about an object from the KG, and the *Natural Language Question Generator*, which is used as an encoder and decoder of the object's representation. Since this model has been able to outperform comparable approaches on question generation, we could consider adapting some of the assumptions, such as deriving the triple relations of the KG, for our project implementation.

Perhaps the most widely-known implementation of the 20Q game is that of Burgener (2006), which can be found at `20q.net` and is also a popular toy. With more than 88 million plays, Burgener's implementation has a precision rate of 80% when it asks twenty questions, and 95% for twenty-five questions. The patent for the game describes the implementation of the deep ANN, which is structured as a matrix of target objects by questions. Each cell of the matrix contains an input-output connection weight, which defines the relationship between the questions/answers and the target objects. The network has two so-called modes: the first takes questions as input and targets as output, and the second takes target objects as input nodes and questions as outputs. The first mode maps answers to weights, while the second mode ranks questions. Similar to Dey et al. (2019), target objects are prioritized, rather than filtered. This is a primary motivation for Burgener's choice of architecture, as it allows the model to correctly predict the target even when given incorrect or inconsistent answers, as mentioned above. As Burgener explains, it also allows the system to take into account cultural/demographic differences that may result in inconsistent answers about a given target object. This is a consideration we should also be mindful of in our implementation, as our proposed model operates on the assumption that the user is providing truthful answers (and that inter-user agreement would be high). The system of weights also allows for a more complex set of inputs than binary yes/no (e.g. sometimes, maybe, depends, rarely) where the degree of certainty of the answer is reflected in the weights used. While our immediate plan is to implement a binary answering system (or perhaps a three-way system, where "unsure/unknown" could also be an answer that would not contribute to the machine's prediction) as a proof of concept, keeping degrees of certainty in mind may help us to improve our final implementation.

Finally, Tonin et al. (2018) describe another ANN implementation of the 20Q framework as

part of a brain-computer interface to enable people with motor impairments to communicate. The system uses a weight matrix to store the strength of the connection between target statements and questions (where negative weights indicate that the expected answer to the question is no, and vice-versa for positive weights). After 15 questions, the model checks to see if there is only one target statement with a positive value. If there is no single positive value after twenty questions, the network returns the statement with the highest current value. When the network correctly estimates the target, the weight matrix is updated. We may want to adopt this sort of "early checking" threshold before a final guess after twenty questions. This paper also presents a very interesting example of how a 20Q implementation may have useful applications outside of the realm of games and entertainment.

2.2 Rule-based Question Generation

In the field of QG, one of the approaches to generating syntactically coherent questions is based on finding rules. In our project, we would like to follow this rule-based approach, most likely using the spaCy[2] library, as we saw in class. This is because questions in 20Q are syntactically limited, so we do not need more complex approaches for this task.

An interesting approach to rule-based QG is the work of Mhatre et al. (2019), which is based on keyword modelling using Named Entity Recognition (NER) to generate questions from an input sequence. Each input sentence is preprocessed and parsed to resolve anaphoric reference. Thereafter, NER is used to identify the type of entity, which is important for the choice of *wh*-component for the QG part of the model. Depending on the output of the NER procedure, the appropriate *wh*-pronoun is chosen. One type of questions they create are yes-no questions, which is the type of question we will focus on. To construct this kind of question, the authors simply perform subject-auxiliary inversion. Since this work makes use of the spaCy architecture, it is of particular interest for this project. In the case of NER, spaCy is a widely used structure that provides a large range of entity types.

Khullar et al. (2018) concentrate on rule-based QG using relative pronouns to achieve high syntactic accuracy and semantic suitability. Their system uses the spaCy dependency parser to evaluate the syntactic structure of sentences. Firstly, the input sentence is parsed to gain information about the presence of relative pronouns and about several linguistic features. Afterwards, this information is input to one rule within a predefined rule set to create the questions. The correct *wh*-component is then determined according to these rules, resulting in a syntactically coherent question.

Both of the above-mentioned systems are fascinating in that they consist of a simple structure by using spaCy's dependency parser and NER methods. By further elaborating on these approaches, we should be able to construct simple yet appropriate yes-no questions for our 20Q model.

[2] https://spacy.io

animal	have_hair	feathers	produce_eggs
aardvark	1	0	0
antelope	1	0	0
badger	1	0	0
bass	0	0	1
bat	1	0	0

Table 1: The first five rows (instances) and three columns (features) in our knowledge base

3 The knowledge base

Our game uses a tabular knowledge base that contains 152 animals and 63 features for each object. Each cell in the table is populated with a 1 or a 0 to indicate whether the given animal does or does not have the given feature.

We are currently developing the implementation on a preliminary knowledge base available here. It is a table consisting of 100 objects (mostly animals) and 28 features for each, and each cell in the table is populated with a 1 or a 0 to indicate that the given animal does or does not have the given feature. For example, a subset of the knowledge base looks like this:

We compiled this dataset by combining a knowledge base with 100 animals and 50 features made available on GitHub with ???.

[WELLESLEY: details about knowledge base construction]

4 Implementation

We opted to implement a relatively deterministic decision-tree-based 20Q system, rather than a deep learning or reinforcement learning paradigm. This was so that the inner workings of our system would remain transparent and everybody would be able to understand the code and work on improving

it. For those of us with an exclusively linguistic background, such a deep dive into deep learning or reinforcement learning felt like too much too soon. This naturally means that our system is less sophisticated than it could have been if we had used a more modern paradigm, but on the other hand, its inner workings are also much easier to understand. In the current section, we outline how we implemented our 20Q system (and see Section 6 for some limitations of the chosen approaches).

4.1 Feature selection

The first question to tackle is "How will the system decide which feature to ask about at each step?" The basic motivation is to choose the most informative feature at each step, so that we narrow down the potential animals most quickly. (We add some non-determinism later, so that it is not always the *most* informative feature that is chosen at each step—this would make for boring gameplay—but first we focus on the basic mechanism for determining the informativity of a feature.)

We used a model in the style of a decision tree (DT) to tell us which feature is most informative at any given step. A DT is "defined by recursively partitioning the input space, and defining a local model in each resulting region of input space" (Murphy, 2012, 545). In our case, the input space consists of the knowledge base described above, and this knowledge base is recursively partitioned by each successive question that the system asks and the user answers.

In standard classification DTs, the space is split by the feature that minimises the entropy (i.e. maximises the information gain; Quinlan 1986) in each partition. However, that method is not applicable here. That method requires an $n:1$ mapping of instances to each class, which is the usual set-up in classification problems: one class contains multiple instances. In our task, though, each individual object equates to a class (i.e. there's an `aardvark` class, an `antelope` class, and so on), so there is only one instance per class (but see Section 4.4 below for handling of non-guessed animals that are not in the knowledge base). The structure of our data makes our problem a non-typical classification task, so a different method must be used.

We chose to orient ourselves around the size of the two partitions of the input space that result from splitting on a given feature, and we select the feature that produces the partitions that are closest to each other in size. To illustrate, say that we split on the first feature given in Table 1 above, `have_hair`. We would end up with one partition containing X animals that have no hair (i.e. where `have_hair` $= 0$), and another partition containing Y animals with hair (i.e. where `have_hair` $= 1$).

To see how even this split is, we take a ratio of these two numbers. We want this ratio to be as close to 1 as possible, since that would represent a perfect split of our input space in half: $\frac{50}{50} = 1$. This is because since we only have two values for each feature (yes or no, 1 or 0), consistently splitting our input space in half results in the highest information gain (cf. Quinlan, 1986; Bishop, 2006).

So, for the feature `have_hair`, we get

$$\frac{|\texttt{have\_hair} = 0|}{|\texttt{have\_hair} = 1|} = \frac{X}{Y} \approx Z.$$

We call this value, Z, the SPLIT CARDINALITY RATIO (SCR) for the feature `have_hair`. In general, the SCR for a feature f is defined as in Equation 1.

$$SCR(f) = \frac{|f = 0|}{|f = 1|} \tag{1}$$

The best feature f_{best} out of all features f is the one for which the distance of the SCR from 1, i.e. $abs(1 - SCR(f))$, is closest to zero. Formally, it is the solution to Equation 2.

$$f_{best} = \underset{f}{\operatorname{argmin}}\ abs(1 - SCR(f)) \tag{2}$$

However, if we were to always choose f_{best} to ask about at every step, the gameplay would be rigid and repetitive. To make for a more varied gameplay, we decided to randomly sample a feature to ask about in proportion to its $abs(1 - SCR(f))$ score. This meant transforming the distribution over f of $abs(1 - SCR(f))$ into a probability distribution, such that the features with the lowest $abs(1 - SCR(f))$ score would have the highest probability of being chosen at each step, but other reasonably informative features are also possible candidates. [ELIZABETH: more detail.]

This method does have certain drawbacks (see Section 6), but in addition to its simplicity, it also comes with one further advantage. Even from the small subset of features shown in Table 1, it is apparent that the features are not independent. For example, if you know that an animal has hair (i.e. `have_hair` $= 1$), you probably also know that it

does not produce eggs (i.e. `produce_eggs` $= 0$). So, an intelligent system should not ask about both of these features, since it should have gained the knowledge contained in both of them by asking about only one. And, indeed, because of the way we bisect the database based on the features that best distinguish the animals still contained in it, we automatically avoid asking about highly correlated features.

For example, let us pretend that the five animals and three features in Table 1 constitute the entire knoweldge base. If we tell the system `have_hair` $= 1$, then it bisects the database and removes all animals with `have_hair` $= 0$. However, since all animals with `have_hair` $= 0$ also have `produce_eggs` $= 1$, the feature `produce_eggs`now also only contains the value 0. This makes it unsuitable as a feature to distinguish animals, so it will not be asked about in subsequent rounds. (Even if two features are not perfectly correlated like these two are, after bisecting the data on one of them, the mixture of 0s and 1s in the other feature will be less even, so it will be less likely to be selected as a feature to ask about.)

4.2 Guessing the animals

Eventually, the gameplay will reach a point when the remaining animals in the pared-down knowledge base can no longer be distinguished by any of the features, since all the features will contain all 0s or all 1s. At this point, the game moves on to the stage of guessing the animals that the user may be thinking of.

We were inspired by the approaches of Dey et al. (2019), Hu et al. (2018), and Burgener (2006), who all maintain a probability distribution over outcomes that is updated over the course of the game based on the answers that the user provides. The motivation behind this approach, as we mentioned above, is to improve the flexibility of the system, and in particular to not reject outright any outcomes that are incompatible with the user's answers. Instead of removing these from the pool of potential answers immediately, their probability is simply reduced, but this means that they are still "in the running" to be guessed by our system, after it has guessed the higher-probability items.

Technically, the distribution that we maintain over animals is not a probability, since it does not sum to one; rather, it is a probability score that keeps track of how (in)compatible each animal is with the answers that the user has provided so far. It can be thought of as bargain-basement Bayesian updating.

We initialise a uniform prior probability score of 20 (an arbitrary choice) across all animals at the beginning of the game and update it after each question based on the user's answers. Specifically, we divide the current probability for each animal in half (also an arbitrary choice) if it is incompatible with the answer the user just gave. Once the features have all been used up, we have a distribution of probability scores across animals that reflect how compatible they are with the user's answers.

The animals that are perfectly compatible with everything the user has entered have the same probability score that they started with; those that are incompatible with only one answer have half of the prior score, and so on. Figure 1 illustrates how this distribution changes over the course of a game. At the end of the game, the system guesses the animals in decreasing order of probability. For the example shown in Figure 1, it would guess `polar bear` and `hippopotamus` (the only ones with a probability score of 20), followed by the animals with a probability score of 10, and so on.

4.3 Rule-based question generation

[ANNA]

4.4 Incorporating out-of-database objects

[RODRIGO]

"the perceived world is not an unstructured total set of equiprobable co-occurring attributes. Rather, the material objects of the world are perceived to possess (in Garner's 1974, sense) high correlational structure. That is, given a knower who perceives the complex attributes of feathers, fur, and wings, it is an empirical fact provided by the perceived world that wings co-occur with feathers more than with fur" (**?**, 29)

The call graph in Figure 2 provides a visual summary of how the functions in the `TwentyQuestions` class fit together.

5 Evaluation

[introductory sentence]

5.1 Win/loss rate

[There is a straightforward measure of success of our system overall: $\frac{\text{games won}}{\text{games played}}$. But because the system will (hopefully) consistently improve as we

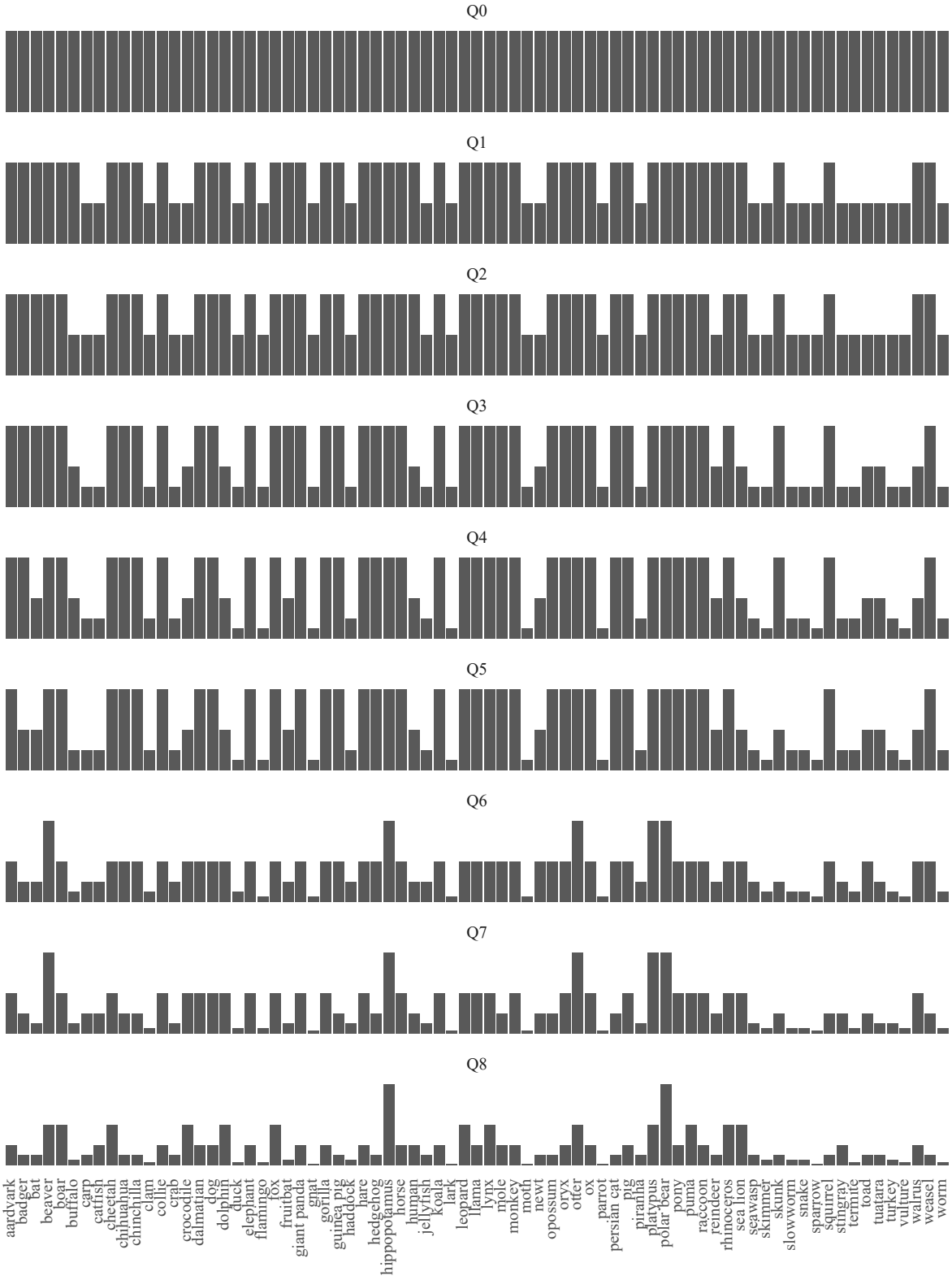


Figure 1: Updating of probability scores across nine questions (only shown for a subset of 75 animals)

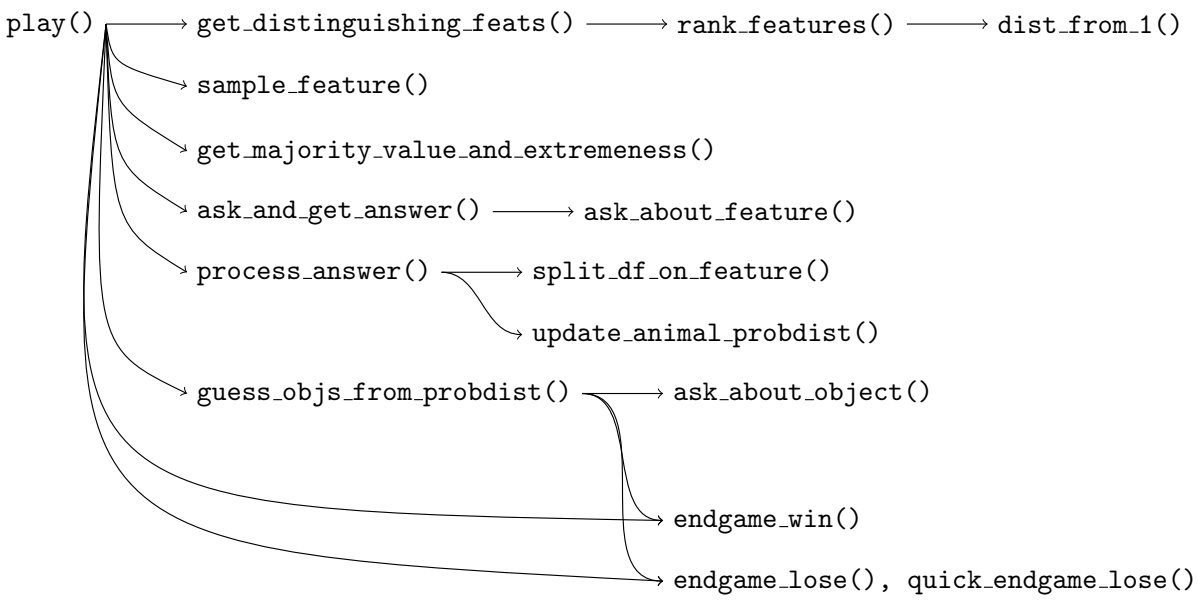
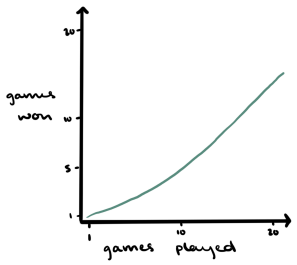
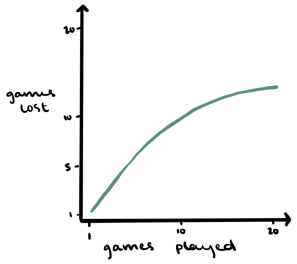


Figure 2: Call graph of class methods in `TwentyQuestions` (WIP ELIZABETH)

add new objects to its knowledge base, the success rates we get in early games with the system will presumably not be as good as success rates from later games. We can track this in a sort of growth curve across games:



The slope of the curve won't ever get bigger than one, but we'd want it to approach one, since that means the system is winning every game that it plays. The slope of that curve at its endpoint is actually given by the rational expression $\frac{\text{games won}}{\text{games played}}$ (Baayen, 2001, 50–51), so we can evaluate how our system performs after some arbitrary number of games by looking at how close that value is to one. Or alternately, we could look at $\frac{\text{games lost}}{\text{games played}}$ and see how close that value is to zero.



Depending on how intense we want to get, we could model this second growth curve using a Zipf-Mandelbrot model (Evert, 2004), and that would allow us to predict how many games our system would have lost after an arbitrarily large number of games played. There is a package in R that we could use to do this: `zipfR` by Baroni and Evert (2014).

(We should also keep track of whether/how many losses come from exceeding the twenty-question limit and how many come from out-of-database items. Ideally, the 20-question-limit curve will not change much, and the out-of-database curve would change more. We should also keep track of the number of questions that it takes for the system to win.)]

5.2 Quality of interpolated out-of-database items

[In addition to evaluating the system's performance in gameplay, we should also evaluate the goodness of the out-of-database items that we interpolate. One way to do this would be to use "crowdworkers" (our friends, probably) to annotate the previously-unseen objects for all of the features that we use, and then compare those results to those output by the interpolation system.]

6 Limitations

- feature selection works by choosing the best feature for the remaining partition of the knowledge base, but if the animal that we're actually looking for does not get retained, then

we don't end up asking questions that lead us to it; rather, we ask questions that are designed to lead us to some animal that is still in the partitioned knowledge base. This is somewhat saved by the probability score distribution across animals, but it is less likely in this case that the correct animal will be guessed.

- Admittedly, this feature selection system does not learn to ask better questions over time; the method of choosing the best feature to ask about at a given time does not follow a strategy that extends beyond each individual turn. We chose to focus our efforts on other parts of the project instead, since the current strategy already works quite well, and automatically deals with correlated features (see below). If we were to develop the system further, though, this would certainly be a point of improvement, e.g. by learning from previous games which sequences of questions tend to perform well.

7 Conclusion

...

References

R. Harald Baayen. 2001. *Word Frequency Distributions*. Kluwer Academic Publishers, Dordrecht/Boston/London.

Marco Baroni and Stefan Evert. 2014. The zipfR package for lexical statistics: A tutorial introduction. Available from `http://zipfr.r-forge.r-project.org/`.

Christopher Bishop. 2006. *Pattern Recognition And Machine Learning*. Springer.

Robin Burgener. 2006. Artificial neural network guessing method and game. US Patent App. 11/102,105.

Alvin Dey, Harsh Kumar Jain, Vikash Kumar Pandey, and Tanmoy Chakraborty. 2019. All it takes is 20 Questions!: A knowledge graph based approach. *arXiv preprint arXiv:1911.05161*.

Stefan Evert. 2004. A simple LNRE model for random character sequences. In *Proceedings of JADT 2004: 7es Journes internationales dAnalyse statistique des Donnes Textuelles*.

Fréderic Godin, Anjishnu Kumar, and Arpit Mittal. 2019. Learning when not to answer: A ternary reward structure for reinforcement learning based question answering. In *Proceedings of NAACL-HLT 2019*, pages 122–129.

Huang Hu, Xianchao Wu, Bingfeng Luo, Chongyang Tao, Can Xu, Wei Wu, and Zhan Chen. 2018. Playing 20 question game with policy-based reinforcement learning. *arXiv preprint arXiv:1808.07645*.

Payal Khullar, Konigari Rachna, Mukul Hase, and Manish Shrivastava. 2018. Automatic question generation using relative pronouns and adverbs. In *Proceedings of ACL 2018, Student Research Workshop*, pages 153–158.

Kaksha Mhatre, Akshada Thube, Hemraj Mahadeshwar, and Avinash Shrivas. 2019. Question generation using NLP. *International Journal of Scientific Research & Engineering Trends*, 5(2):394–397.

Kevin P. Murphy. 2012. *Machine learning: A probabilistic perspective*. MIT Press, Cambridge/London.

J.R. Quinlan. 1986. Induction of decision trees. *Machine Learning*, 1:81–106.

Sathish Reddy, Dinesh Raghu, Mitesh M. Khapra, and Sachindra Joshi. 2017. Generating natural language question-answer pairs from a knowledge graph using a RNN based question generation model. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 376–385, Valencia, Spain. Association for Computational Linguistics.

Alessandro Tonin, Niels Birbaumer, and Ujwal Chaudhary. 2018. A 20-questions-based binary spelling interface for communication systems. *Brain sciences*, 8(7):126.

A Individual contributions

All group members were in regular, active communication about ideas and directions in which to take the project. It was a fully collaborative effort throughout. We summarise here how we divided the workload between the four of us.

Wellesley Literature work, prepared and held the in-class presentation, wrote and proofread sections of the project plan and the final paper.

Anna Literature work, implemented the question generation, wrote sections of the project plan and the final paper.

Rodrigo Implemented the handling of out-of-database items, wrote sections of the final paper.

Elizabeth Implemented the `TwentyQuestions` class and its core methods, wrote sections of the project plan and the final paper, hosted the GitHub repo, compiled LaTeX files.