Project plan: 20 Questions

Wellesley Boboc, Anna-Janina Goecke, Rodrigo Lopez Portillo Alcocer, Elizabeth Pankratz

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. There, 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.

Related work

Previous 20Q Implementations: Knowledge Graphs, Reinforcement Learning, Artificial Neural Networks

Past approaches to the implementation of a 20Q system have made use of Knowledge Graphs (KG) in combination with rule-based question generation (QG; Dey et al. 2019), Reinforcement Learning (Hu et al. 2018), or variations of Artificial Neural Networks (ANN) (Burgener 2006, Tonin, Birbaumer & Chaudhary 2018). Below we will discuss the relative merits of these implementations and how they inform our work on this project.

Jedynak, Frazier & Sznitman (2012) describe a 20Q implementation with noise, discussing Bayes optimal policies for entropy loss. Their model seeks to minimize, after a fixed number of questions, the expected entropy of the posterior by implementing a probabilistic bisection strategy (Horstein 1963, Waeber, Frazier & Henderson 2011), and

discusses uses of this method for stochastic search, computer vision, and other applications.

Dey et al. (2019) implemented another instance of a probabilistic model, one 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 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. 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 KG to create 20Q 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.

Reddy et al. (2017) propose the application of KG to generate sets of question-answer pairs within a Recurrent Neural Network architecture by deriving triple relations from given entities. The triples are composed by a subject, an object (both are represented as nodes in the KG) and a predicate (represented as 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 would consider refining some of the assumptions, such as deriving a triple relations of the KG, for our project implementation.

Hu et al. (2018) implement a policy-based system of 20Q that uses reinforcement throughout the game. While an entropy-based method makes use of a KG, in this work, the model for selecting questions is based on Reinforcement Learning procedures trying to find the optimal reward function. The questioner agent relies on a probability distribution over all objects which is then updated according to the answers. Hu et al. (2018) suggest a RewardNet to learn the immediate reward at each time step to improve the overall performance of the model. The model continually improved its win rate over time and was shown to be able to identify the target object within 14 questions.

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 20 questions, and 95% for 25 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 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 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 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), keeping degrees of certainty in mind may help us to improve our final implementation.

Tonin, Birbaumer & Chaudhary (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 20 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 20 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.

Rule-based Question Generation

In the field of Natural Language Question Generation, one of the approaches to generate syntactically coherent questions is based on finding rules. The work of Mhatre et al.

(2019), for instance, 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. To construct this kind of question, Mhatre et al. (2019) perform subject-auxiliary inversion. Since this work makes use of the spaCy architecture (https://spacy.io), it is of particular interest for this project. We would like to follow a rule-based approach regarding the generation of questions. 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 these approaches we should be able to construct simple yet appropriate yes-no questions for our 20Q model.

Data

We are currently developing the implementation on a preliminary knowledge set 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, the first few rows and columns of the dataset look like this:

Whichever dataset we end up actually using should be larger than this and have many more features, since the system is currently almost always able to finish within the 20-question limit. We also found another promising dataset called Animals with Attributesconsisting of around 50 classes and 85 attributes. We are currently looking for larger open-source knowledge bases like this to use.

Implementation

The machine learning system that we will use to implement this task is fairly simple: a decision tree (DT), also known as a CART model, which stands for "classification and regression tree" (Murphy 2012: Section 16.2). 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. However, in contrast to defining a local model in *each* resulting region of input space, we will only retain the subset of the knowledge base that is compatible with the user's answers.

But how will we partition the space in an optimal way, or in other words, how will we choose which question to ask? One standard method in classification DTs splits the space on 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. That makes our problem a non-typical classification task, so a different method needs to 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 above, *Hair*. We would end up with one partition containing 57 animals that have no hair (i.e. where $Hair = 0$), and another partition containing 43 animals with hair (i.e. where $Hair = 1$). 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 "branches" for each feature (yes or no, 1 or 0), consistently splitting our input space in half would result in the highest information gain regarding the class label. We selected this approach based on tree-based algorithms such as ID3 (Quinlan 1986, Bishop 2006).

So, for the feature *Hair*, we get

$$\frac{|Hair = 0|}{|Hair = 1|} = \frac{57}{43} \approx 1.33.$$

We call this value, 1.33, the SPLIT CARDINALITY RATIO (SCR) for the feature *Hair*, and we want to choose the feature whose SCR is as close as possible to 1. Or in other words, we want to choose the feature f for which $abs(1 - SCR(f))$ is minimised. We illustrate

this principle by also looking at the feature that actually minimises $abs(1 - SCR(f))$ in the current development dataset, *Predator*:

For *Hair*, this value is $abs(1 - 1.33) = 0.33$, which is greater than 0.18, so given the choice between these two features, our system would choose to split the input space on *Predator* rather than *Hair*. In question-processing terms, this means we would ask the user a question like "Is it a predator?" rather than "Does it have hair?".

So in this way, for each question the system has to ask, we compute the SCRs of all features in the input space, and choose the one with the smallest value of $abs(1 - SCR)$ (or if there are multiple features with the same value, we randomly choose between them). The first split of the development dataset is always on *Predator*, since that results in the most even partition of the input space into subsets of size 45 or 55, depending on whether the user answers 0 or 1.

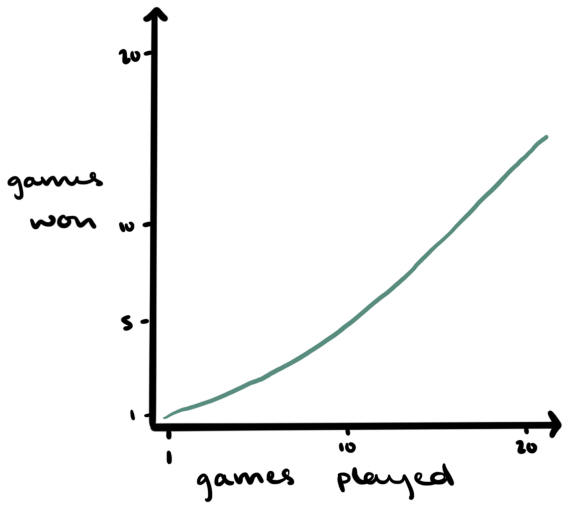
There are certainly other strategies that could be used for choosing a question. This algorithm could be made more sophisticated by learning from previous games, or by performing an analysis of the features to see which ones are mutually exclusive, and so on. But we're keeping it simple for now.

The steps in our implementation will be as follows (everything following 1 can be concurrent):

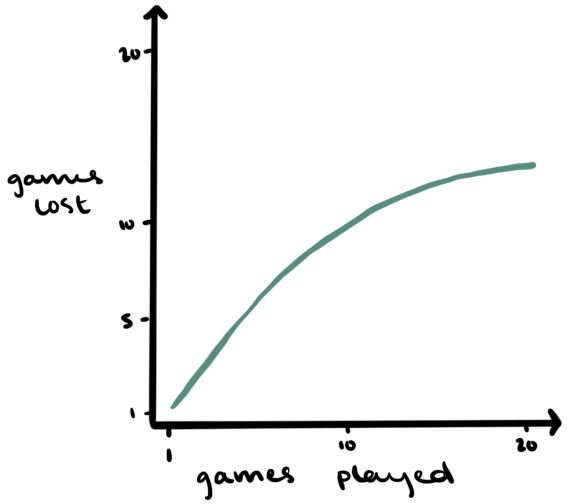
1. Write baseline DT system that decides at each step the optimal question to ask, narrows down the knowledge base to the objects compatible with the yes/no answers it receives, and ultimately makes guesses until the number of questions has reached 20 (we've pretty much finished this already).
2. Incorporate handling of answers beyond yes and no.
3. Implement the rule-based question generation, based on the structures of yes/no questions in corpora, e.g. Switchboard.
4. Implement the incorporation of previously-unseen objects into the knowledge base.

Evaluation

There is a straightforward measure of success of our system overall: $\frac{\text{games won}}{\text{games played}}$. But because the system will consistently improve as we 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 x 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 about this whole thing, 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 & Evert (2014).

(We should also keep track of whether/how many losses come from exceeding the 20 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.

Wellesley also mentioned that we should count the number of questions that it takes for the system to win.)

Also, we should evaluate the goodness of the out-of-database items that we interpolate. We could use human annotators who don't look at any of the other items in the dataset and only annotate the out-of-database items for all the features we use? Then we can compare their results to those output by Rodrigo's system. (We could either do it ourselves or source some "crowdworkers" among our friends using Google Forms again—will leave this up to Anna and/or Wellesley).

References

Baayen, R. Harald. 2001. *Word frequency distributions*. Dordrecht/Boston/London: Kluwer Academic Publishers.

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

Bishop, Christopher. 2006. *Pattern recognition and machine learning*. Springer.

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

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

Evert, Stefan. 2004. A simple LNRE model for random character sequences. In *Proceedings of JADT 2004: 7es Journées internationales d'Analyse statistique des Données Textuelles*.

Horstein, M. 1963. Sequential transmission using noiseless feedback. *IEEE Transactions on Information Theory* 9(3). 136–143.

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

Jedynak, Bruno, Peter I. Frazier & Raphael Sznitman. 2012. Twenty questions with noise: bayes optimal policies for entropy loss. *Journal of Applied Probability* 49(1). 114–136.

Khullar, Payal, Konigari Rachna, Mukul Hase & Manish Shrivastava. 2018. Automatic question generation using relative pronouns and adverbs. In *Proceedings of acl 2018, student research workshop*, 153–158.

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

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

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

Reddy, Sathish, Dinesh Raghu, Mitesh M. Khapra & 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*, 376–385. Valencia, Spain: Association for Computational Linguistics. `https://www.aclweb.org/anthology/E17-1036`.

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

Waeber, Rolf, Peter I. Frazier & Shane G. Henderson. 2011. A Bayesian approach to stochastic root finding. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, 4033–4045.